A-PDF MERGER DEMO

USER–FRIENDLY INTERACTION WITH DATA AS EXEMPLIFIED USING PHARMACOLOGICAL PACKAGE–INSERT TEXTBASES

Alvin Joseph Varughese

A research project submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, 2006.

DECLARATION

I declare that this research report is my own, unaided work. It is being submitted for the Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before any degree or examination in any other University.

_____

(Signature of Candidate)

_____ day of _____ (year) _____

ABSTRACT

The Medical field is vast and dynamic. To be effective in their professions, doctors are required to keep track of the constant growth in knowledge that is taking place, although it is unrealistic to expect them to do so fully. The knowledge in this field is inherently descriptive and thus an unlikely candidate for storage in traditional databases, but instead are found in textbases. In spite of the limitations of databases, they are still used in the medical profession as they remain the standard means of storing information. The aim of this study was to demonstrate that it is possible to provide a single interface to medical knowledge in its more natural form, a textbase. This was achieved by investigating the relevant areas, creating models for the various elements of the system, and subsequently constructing such a system for demonstration. Through the use of data interaction models, this work shows that the doctor's knowledge can be adequately supplemented, as validated by testing, enabling them to make correct decisions using the information provided. Thus the opportunity exists to implement the methodology in other areas where descriptive knowledge exists.

In dedication to my God
for giving me life abundant
and
to my Parents
for the love and dedication
they have shown in raising and guiding me.

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Dr. Ian Kennedy and Dr. David Rubin whose guidance and their insightful comments which helped shape this work and to my brother, Melvin Varughese who proof read this work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PAPER

User-friendly interaction with data as exemplified using pharmacological package-insert textbases

Alvin Varughese

Abstract— The Medical field is vast and dynamic. To be effective in their professions, doctors are required to keep track of the constant growth in knowledge that is taking place, although it is unrealistic to expect them to do so fully. The knowledge in this field is inherently descriptive and thus an unlikely candidate for storage in traditional databases, but instead are found in textbases. In spite of the limitations of databases, they are still used in the medical profession as they remain the standard means of storing information. The aim of this study was to demonstrate that it is possible to provide a single interface to medical knowledge in its more natural form, a textbase. This was achieved by investigating the relevant areas, creating models for the various elements of the system, and subsequently constructing such a system for demonstration. Through the use of data interaction models, this work shows that the doctor's knowledge can be adequately supplemented, as validated by testing, enabling them to make correct decisions using the information provided. Thus the opportunity exists to implement the methodology in other areas where descriptive knowledge exists.

***Index Terms*— Data Interaction, Pharmacology.**

I. Introduction

THE Internet and its resultant popularity has led to much information being made available from many kinds of sources. Although the Internet has allowed information to be shared, it has its downfalls as there is no regulation or review process that dictates the kind of information that is posted. Another key aspect of the Internet is that it is dynamic. It is constantly growing and changing and thus the search tools themselves are expected to be dynamic, to keep track of where the best-practice information is.

Medical knowledge has progressed greatly over the last few decades, and many diseases have been eradicated through the development of a host of vaccines and drugs. As much of this development occurred in First World countries with multi-million dollar programs, many of the drugs produced are priced beyond the reach of the average third world person. This led to many pharmaceutical companies providing generic alternatives to the initially developed drugs. Hence, many medical practitioners have to continually keep abreast of the developments in generic medicines and also know a host of drugs that can bring relief to suffering patients.

Occasionally, patients suffer premature death from incorrect medication or doses being prescribed [1]. What has become clear is that some of these deaths could have been avoided if relevant and correct information is made available at the time of prescription. It is thus vital to provide doctors with reviewed documentation on how to use the medication, its indications, and the doses and reactions that may result. Traditionally the medical profession has come to value the use of medical databases to maintain information stores which are accessed as necessary by the professionals. Databases have a particular structure which allows them to be maintained and added to.

However, medical knowledge tends to be descriptive in nature whereby every word contributes to the understanding. This is because illnesses are often described by symptoms, signs and diagnostic test results which by their very nature must be descriptive. As medical knowledge is **descriptive** it is not sufficiently well structured that one can place the information into records and fields suitable for incorporation into a database. Also, when knowledge is descriptive, there is a possibility of it being verbose. Finally, descriptions use an uncontrolled format and vocabulary, which would require significant processing to retrieve relevant information. Retrieval of descriptive text from a textbase is thus more difficult than retrieving records from a database.

Therefore, it becomes clear that by making the target of the search a *word* as opposed to a database *field*, information retrieved from a textbase would be more relevant to queries posted by a medical professional. Despite the inherent difficulties posed by retrieval of information from descriptive text, as described above, the study aims to demonstrate that this is nonetheless possible. To do so, it was necessary to investigate the relevant areas, develop an appropriate approach with the hindsight of previous related research, construct an adequate software prototype and validate it through testing focused on predetermined goals.

The author is with the School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg, South Africa 2050 (phone: +27 83 451 2797; fax: +27 11 475 6992; e-mail: alvinjv@yahoo.com).

II. Review of Literature

A. Computer Aided Medical Diagnosis

The volume of medical knowledge is such that today no doctor can access or memorise all the necessary information in their daily practice.

There are a number of proponents who believe that use of computers in diagnosis will aid professionals in their work. In response to this belief, the French developed a system that would allow medical professionals access to large amounts of information through a knowledge-base system known as ADM [2]. They demonstrated that by developing a Web interface to their internal database of medical information it could be a useful supplement to the doctor's knowledge, by providing results in a shorter period of time. "Hypermedia document allows navigation through descriptions and diseases lists easily and quickly without losing context." [2] Although their research proved to be successful in that they were able to provide a widely accessible interface to a locally stored database, a lot more information exists on the Web which is inaccessible through their system. Thus research was needed to develop a system that provides a single interface that could cater for the multitude of sources of medical information available on the Web. This would ensure that the limitations in their system are overcome.

On a different front, drug development incorporates an information intensive process of populating databases which include physical property databases, pharmacological effect databases, bioactivity sources and text reports. To effectively carry out the drug prescription process it is necessary to sort relevant information from all the available data. Since drug prescription is a vital process for the doctor, it is necessary to find an information system that will point to possible prescription problems before they are encountered. For this reason, there is a need to have a single interface to the disparate sources of information that are available in the pharmaceutical environment. Advanced search engines create a single point of access to internal and external information [3].

While it was proposed that search engines will provide the necessary information, no strides have been made towards providing or testing the viability of a solution. This research provided some findings that proved to be relevant to the current research as they both have the same, pharmaceutical, problem domain. This is because they are the same problem, with slightly different details. It was proposed that possibly the only way to link up the disparate sources of pharmaceutical information is through advanced search engines, which thus points towards the use of search engines as opposed to traditional query-language databases [3].

B. Symptoms Diagnosis

It is important to realise that the process of diagnosis goes beyond the mere provision of medical information. There are many sites where medical information is readily accessible to any user. However the Website creators are quick to point out that it is has a limited diagnostic utility. *"Diagnosis takes place in the sealed black box of the physician's mind. As far as anything is known of its algorithms, diagnosis is based on pattern recognition using incomplete and often inaccurate information. It is informed by the clinician's idiosyncratic perception of the probabilities of conditions."* [4]

This knowledge served to limit the scope of the study at an early stage as it helped clarify the well known fact that the process of diagnosis could not easily be modelled or automated. This is important to the research as it makes it quite clear that whatever system is finally produced, it must rely on **the doctor searching for the right things** and **the system finding the best possible results** for those searches. The results of these searches do not provide the diagnosis. **Rather, the aim of the system is to aid diagnosis and then present treatment options.**

Other sites provide different perspectives that are useful. Some provide useful ways of presenting the results by grouping results according to illnesses that may cause the specified symptoms [5]. In this way, doctors can quickly cross off one of the possibilities as they go through the groupings. This also was of benefit to the current research as it provided another way of presenting results to the user which may improve diagnosis. In reading through relevant texts on diagnosis, it quickly became apparent that **a doctor will best be aided by a system which *speedily* and *efficiently* provides relevant knowledge.**

C. Pharmaceutical Databases

Along the same lines, pharmaceutical databases have been set up to inform users of the appropriate use of drugs. The US Food and Drug Administration Agency (FDA) is the primary approvals board for new drugs being developed. Hence they have an extensive document base of approved drugs and the tests performed in approving the drugs. It is interesting to note that their Web page is powered by Google [6]. The results use Google's PageRank algorithm and results seem to be retrieved from a local hard drive which might indicate the use of the Google Desktop. It proved to be a further indication of the use of search engines in the pharmaceutical industry to retrieve results based on user queries.

Given the large amount of information that can be associated with any particular drug, some sites focus on providing all that information to the benefit of both practitioners and other users. Information such as interactions, side effects, delivery detail, diseases and symptoms related to a particular drug are displayed and clustered accordingly [7]. By clustering the information it becomes more than just comprehensive, its components also become components easily accessible to the doctor.

D. Database Searches

Databases provide the content storage for many sites, which dynamically create Web pages around them, including commercial catalogue sites, online news, and even

entertainment sites. Intranets often contain large amounts of text stored in databases as well. Some authors point out that even though many databases have search functions, they are not oriented towards text searching. “Database search is not oriented towards text search and relevance ranking: it is great for locating widgets by part number and listing the inventory of leather slippers, but not so good at helping site visitors find the articles on widget quality or comparing leather and fleece slippers.” [8]

Suggestions have been made about using text search indexers with databases to make up for the above inadequacies. Text indexers achieve this by making use of indexes to keep track of where particular words exist in a database, thus allowing the word to be searchable. These suggestions do point to the need to be able to text-index content to effectively search in it, a fact that is particularly relevant in providing pharmaceutical information.

There are many online databases of research papers that contain much information on a wide variety of topics including the pharmaceutical field. To retrieve relevant data, it is necessary to perform a search so that only relevant papers are retrieved. There is substantial variability in the interface and search software for these databases. However, there are common skills that are useful in the efficient retrieval of sufficient yet relevant information [9].

Although databases with text search indexers are capable of providing text searching capabilities, it still remains difficult to recompose medical information into the fields required by a database. Hence the use of textbases remains the desired approach. Unfortunately, little literature exists showing a comparison between traditional databases and textbases, particularly in the pharmaceutical field.

E. Using Search Engines

Internet search engines are a class of sites on the Web that are designed to help people find information stored on other sites [10]. Different search engines use slightly different methods to find results and **these results can be re-processed to produce a more relevant set of results.** This realization is relevant to the proposed research as it provides significant information about **the way queries should be constructed** to retrieve the most relevant, and controlled, information for the data-interaction model to process.

F. Textbases versus Databases

Having looked at both textbases and databases, the next logical progression is to compare the two methods of storing and retrieving information. The Internet gave users around the world access to large amounts of unorganised, unstructured information. In many ways this led to much more knowledge being shared. However, with no standards dictating the way information was put onto this network, there was no structure. Hence, the way this data has to be searched had to be different to traditional database searches. It is necessary to understand how searches of textbases and databases work to effectively perform a search. When using any search engine, the difference between this technology and structured database technology must be understood to avoid unrealistic expectations [11].

G. Desktop Search Engines

The current work investigated the possibility of using Desktop search engines to interface to a locally maintained data store of relevant medical information. Some authors compare the features that Google Desktop has versus that of MSN Desktop. The features differ due to the different approaches that the two Desktop Search engines take in performing a search. It is evident that the two engines have different strengths [12].

Obvious limitations are that the features being compared may be only relevant for the *average* user and thus other features may have to be tested and analysed for relevance to the medical specialist. However, the analysis still remains valid as it possibly identifies the deciding factor in choosing MSN or Google's product. To provide an efficient single interface to disparate sources of data requires an understanding of how the search must be formed to get relevant pharmaceutical data.

Other authors believe that MSN Desktop is a far better product than Google Desktop [13]. The two products were released within a relatively short space of time of each other, which led to comparisons being made. Some authors maintain that MSN Desktop must be a better program given its close links with the operating system and the more relevant criteria for search [13]. In the end, pragmatism may have to prevail. While this may be relevant and valid, it may be the product of a one dimensional view of the way searches may need to be performed.

H. Quality and Relevance of Information

The most significant aspects of the current research were the methods that had to be developed to retrieve, in this project, the most relevant data for a doctor from documented knowledge. This is in support of the fundamental question of **why a doctor would need to perform such a search**. Information need is classified generally as follows:

- Informational, to find specific content on a site.
- Navigational, where intent is to reach a site.
- Transactional, where intent is to perform an activity over the Internet.

Classic information retrieval is inherently based on users searching for information — the so called “information need”. But the need behind a Web search is often not informational; it might be navigational or transactional. By taking these various goals of users into account when performing the searches, far more relevant results can be produced. For a successful Web search, it is essential to understand this classification [14].

Studies have been conducted to understand how people search and what they are searching for, but not *why* they are searching. The *why* is the essence of understanding Web behaviour. This is more important to the task of retrieving results than has been previously accepted. “More

importantly, an understanding of search goals provides a foundation for tackling the larger problems of conveying user goals to a search engine… and modifying the engines' algorithms and interfaces to exploit this knowledge." [15]

Research and business is currently moving from centralized databases towards the incorporation of disparate information sources. This is a reflection of the way information sharing has evolved in the Information age. While connecting these disparate sources of information remains a major goal, the *quality* is an important issue for large-scale integrated information systems [16].

III. Interacting with Information

A. Understanding the need for information

The Information Age has brought on what its name would suggest — much information. Though there is an abundance of information, it would seem that pressures of time leave many professionals with less of it to absorb that abundance into their decision-making processes. **For information to become useful it must be:**

- **relevant,**
- **sufficient and not excessive,**
- **retrieved quickly.**

With the above in mind it became apparent that there is a need to quickly provide useful information in the medical profession. No single person is capable of holding and maintaining all the relevant knowledge even if it is only for their specific field of medicine. This problem is further compounded by the continual development of new drugs and their generic alternatives which the doctor may need to prescribe to treat illnesses. Many attempts have been made to alleviate these problems through the creation of medical databases such as the ADM [2]. Thus far, such resources have proved to be limited as more pharmacological information becomes available from around the world. For any such system to function effectively requires that a single interface to these disparate sources of information be created.

B. Mapping a new data interaction approach

The Internet showed that knowledge could rapidly be shared all around the world. Thus far the benefits that have been reaped have been limited due to the inherent problems in that anyone is allowed to post any kind of information. Given a lack of regulatory or review process, much of the content on the Internet is irrelevant or worse still, incorrect. Hence the Web, which might be a useful supplement to a doctor's knowledge-base, has justifiably been set aside for requiring too much effort to be spent in verifying the authenticity of such information. What the Web did show, was that in order **to effectively search for information that was descriptive, every word held in every document had to become of value and thus be searchable.**

Many believe that to supplement a doctor's knowledge adequately requires that a controlled database of information be set up [2]. Databases have often been used to represent information into a structured form such that they can easily be searched. However there are numerous forms of information, particularly descriptive information, where databases, in the traditional sense, are no longer suitable. This is primarily because the information must by necessity be *structured* for it to be placed into a database, something that is not easily done with descriptive information. Medical conditions and their treatments are *descriptive* in nature and hence hyper-textual documents are more suited to containing relevant knowledge. Thus a medical textbase would by necessity be composed of such documents. Through controlling the kind of documentation that is entered into the textbase, the veracity of the information can be ensured.

As was alluded to in the literature review, no work has compared the effectiveness of textbases versus databases. Even though this would be of interest, it was felt at a very early stage the primary aim should be to prove the effectiveness of the textbase system, given the many benefits that could be reaped by its use in descriptive fields. Hence many of the aims of the study were shaped by this.

There are a variety of different methods one can adopt in conducting a search. They vary primarily in what the unit of search is. This could either be a *word* or a *field*. A variety of desktop search engines have been released that make the unit of search a word and thereby allow full indexing of the contents of a local drive. By choosing a document format as the base of the medical knowledge, the searching approach must be adapted for the text format of the content. Time constraints also prevented a more extensive study into whether or not text indexers would provide the required searching capabilities in traditional databases. Furthermore, a more extensive analysis of which desktop search engine would be best proved to be beyond the scope of the primary objectives. Of greater importance was the need to take into account the user's goals in the way the system would eventually be constructed. In doing so it then was possible to ensure that relevant data is presented in response to a doctor's query. Having focused and consolidated the problems presented in the literature survey, the primary aims of the study could be determined.

C. Research Problem

The research problem was to investigate whether a *document* could become the root information source through which a doctor's knowledge could be supplemented. To demonstrate this, valid criteria against which success could be determined had to be generated and thus the following hypotheses were postulated:

1. It is possible to **return relevant documentation in response to a doctor's query**,
2. It is possible to ***speedily*** **and** ***efficiently*** **extract relevant information from a selected document** as per a doctor's needs so as to provide a suitable aid,
3. It is possible to cater for the growing knowledge in the field by **easily interacting with newly added documentation**.

IV. ADOPTING A NEW DATA INTERACTION APPROACH

A. *Extracting a high level overview*

Since there is a need for doctors to interact with their data in a user-friendly way, the current work modelled how such an interaction would take place. The problem with modelling such an interaction was that there are a number of actors that affect the way in which that interaction takes place. Having placed the problem into context it was then necessary to understand the unique dynamics of the problem by breaking it into its smaller entities and identifying the actors. Examining the interactions between the entities and the actors then allowed an understanding of the requirements of those interactions. Furthermore, by understanding those interactions it was then possible to identify levels of functionality which served as milestones in the software development cycle.

Information retrieval naturally has to be dictated by goals. The doctor enters a query for the purpose of finding relevant information. Since not all documents contained in our adopted package-insert data source will have information consistent with the doctor's query, a method of delivering only relevant documentation to the data interaction model was found. This documentation could then be submitted to our Data Interaction Model (DIM) for further processing. Since the end user of the system is the doctor, the primary actor, it was necessary to model the doctor's need for information in the many (*d*) forms it exists, as seen in figure 1. Since the information is contained in documents, which do not have a specific format, it is necessary to model the documents such that it is possible to find information contained therein. The data interaction model can then make use of the models it has for both the doctor and the package insert to provide the wanted content to the interface module. The interface module is then required to organise the data in a way that the doctors can easily interact with the data they have requested.

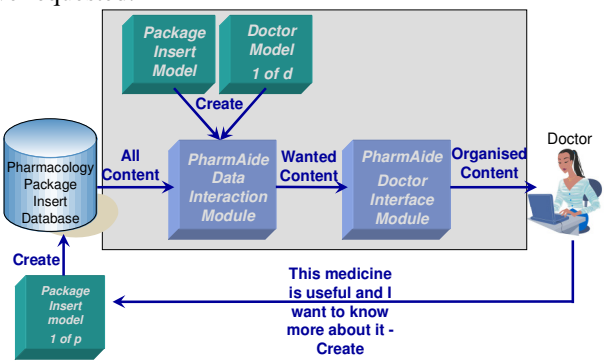

Figure 1. A high level overview of the system

The medical field, like many others, has a body of knowledge that is continually growing. To fully meet the requirements of the previously mentioned problem of the growth of knowledge, it was also necessary to allow for the system to interact with newly added data. As new medications are developed, package inserts, describing the way such medication is used, are printed and inserted into the package with the medicine. These inserts come in different forms and hence the system has to deal with different (*p*) formats of insert as shown in figure 1. For the purposes of this project the formats accepted were be hyper-textual (HTML), Adobe (PDF) documents, and Microsoft Word documents (DOC). Any documents of the aforementioned formats can therefore be added to the database at the doctor's discretion. Then they can be accessed through the Data Interaction Model in response to a doctor's query. The validity of these documents can thus be verified by the doctor before inclusion into the database, thereby gaining the benefits of traditional validated medical databases. These documents then became the body of knowledge that the system interacts with to provide relevant information.

Having extracted the high level overview of the system it was then possible to determine how to best develop the software to meet the identified requirements.

B. *Software development approach*

Given the time frame for implementation of the project and the objectives to be achieved, it was decided that a hybrid Spiral Lifecycle-Rapid prototyping approach would best meet those requirements. Typically a Spiral Lifecycle approach would require the following actions to be taken at every stage of development:

- Determine objectives for next level product.
- Identify alternatives and Resolve Risks.
- Develop and verify next level product.
- Plan next phase.

It was felt during the development of the software that the primary object for the project was to demonstrate the concept. Hence it was anticipated that the final prototype would be at best, a beta release. Having that in mind, rapid prototyping was used. The identifying of alternatives and planning of the next phase resulted from a consideration of database versus textual sources and a subsequent choice from the competing desktop search engines available. Thus the hybrid Spiral Lifecycle-Rapid prototyping approach was reduced to the following actions:

- Determine objectives for next level product.
- Develop and verify next level product.

To determine the modules of functionality that were required, the high level overview was divided according to functions:

- The Professionals Interface Module (PIM) is responsible for providing user-friendly interaction with the information retrieved.
- The Search Engine Module (SEM) is responsible for retrieving relevant documentation from the insert database.
- The Data Interface Module (DIM) is responsible for implementation of the model of the package inserts so that relevant information is extracted from the document.
- The Classification module then implements one model of how a doctor would want to interact with his data

and retrieve succinct information about a particular drug.
- The Generic Retrieval Module (GRM) then implements one model of how a doctor would wish to retrieve generics for a particular drug.

At the outset, four stages of development were identified, compromising the five identified modules that made up the final prototype. Each stage of development was undertaken through the use of Rapid prototyping, whereby the initial requirements of each module were tested as they were implemented. Overall, four different prototypes, each representing a new leg of the Spiral, were released, each displaying the additional functionality of the newly encompassed modules as shown in figures 4 and 5. By using the hybrid Spiral Lifecycle-Rapid prototyping approach, functionality being added was validated against the initial requirements, which allowed for the early identification and avoidance of serious problems in the final prototype. Over time the Doctor Interface Module also changed to allow access to the newly added modules.

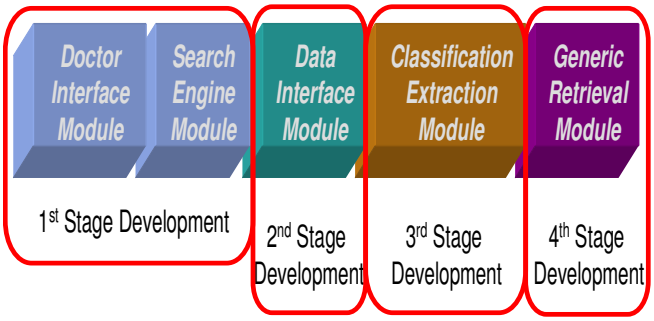

Figure 2. Development stages and the modules they will incorporate in their release

C. *Choosing a Search Engine*

Having made the case for making the unit of the search to be a word when searching documents filled with descriptive knowledge, it becomes necessary to find a tool that will allow that kind of a search to take place. In recent times a number of Desktop search engines have been released allowing users to search content on their hard drives. While the approaches to searching may differ, the features relevant for the purposes of this project make MSN Desktop more suitable. MSN Desktop being a Microsoft product allows for closer ties with the Operating System's filing system.

V. MODELLING THE SYSTEM

Having seen how it is necessary to understand what the user goals are in retrieving information and how they wish to search for it, it was necessary to adopt a development methodology that handled both requirements. It was vitally important to ensure that these goals were identified prior to development.

A. *Modelling the doctor*

The modelling of a doctor was a key aspect of determining the requirements of the system as a whole. As is often the case with software development, an interview was conducted with a general practitioner, the primary actor and end user, to extract high-level requirements for the system. It provided an understanding of how doctors would want to interact with their data. A few of the important insights that came from the initial interview are:

- General practitioners see many patients in a day and are hence pressed for time.
- Traditionally, many would have a South African Medicines Formulary, MIMS, on their desks but finding the required info may be often time consuming.
- Hence, some doctors do not find required information before making a prescription.
- New information is often discarded.

Given the serious effects that maladministration of drugs can have, it thus becomes necessary to ensure that doctors are provided with succinct information from a particular query. To ensure that the information presented is useful it was necessary to model the various forms of doctor's information needs.

1) *Classification extraction model*

There are a number of attributes that a particular drug can have which are of importance to a doctor. So, for example, a national medicine registration number on many occasions serves as nothing but a hindrance to getting at the relevant information that a doctor needs prior to giving a prescription. Hence medication was modelled to have the following **attributes**:

- **Indications**
- **Contra-indications**
- **Side-effects**
- **Dosage**
- **Interactions**
- **Warnings**

The above attributes were then used as key words that were always searched for in any line of text that had been identified as being a heading. If a key word was found, then the associated information was assigned to the attribute matching the key word. Once all the relevant information was extracted from the selected document, the variable of type `Medicine`, as seen in figure 3, is submitted to the Doctor Interface Module for display.

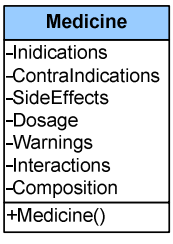

Figure 3. The Medicine Class

2) *Generic retrieval model*

During the initial interview it was also determined that there was another way in which doctors would like to interact with pharmacological data. Generic forms of many drugs have been released. These generics are often cheaper alternatives and at other times may have slightly different compositions. In either case, doctors will find it useful to have generic alternatives to the medication found to match

their query.

As a result, the **composition** of a particular medication is of importance and hence was added to the class as seen in figure 3. Since a composition is defined by one or many quantities, numbers occurring in the composition are extracted as identifiers of weightings in the medication. A search for a non-standard word just prior to or after the identified number would yield the associated components. Hence, a component and its associated weighting could now be used to find generics with the same components and associated weightings. The combination of these is then submitted to the desktop search engine and the results validated to ensure that the search phrase was found in the composition. In doing so, viable alternatives to the initially found medication are presented to the doctor.

B. Modelling the Package Inserts

Documents exist in a number of different formats including hypertextual, Adobe and Word documents. Since different encoding is used to define the contents of such documents it was necessary to decode them into a readable format. By converting these documents into text documents further processing was simplified. For the purposes of this project, only the above three document formats were considered.

Since a number of different companies are responsible for the manufacture of medications and the associated package inserts, the formatting of inserts vary widely. For the model of the package insert to be successful it had to interact with information in the majority of package inserts, in the varying formats found. It was decided pragmatically that the rules implemented to find the required information would only need to be successful 80% of the time. Hence, a basic assumption was that a heading would always be preceded by an empty line. Using that basic assumption, headings were identified and affixed with a marker through the following rules:

- The line is not empty.
- Nor is it a sentence, i.e., it is not suffixed with a '.'.
- Nor is it bracketed.
- Nor are the previous lines null.
- Nor are the preceding 2 lines suffixed with heading markers.

Once a document has been read and the headings marked they were available for later processing.

C. Implementing the models

Having modelled the various actors it was then possible to integrate them with the other entities within the system. Typically, a doctor would enter a query relating to a diagnosis. Given the flexibility that a word based search allows, the search phrase could be for medication or for the particular symptoms that it treats. This query is then passed from the Doctor Interface Module to the Search Engine Module. The Search Engine Module performs the important task of submitting queries to the desktop search engine as well as holding the results returned from the engine. The desktop search engine performs a key role by acting as the first filter to all the available knowledge. It does so by retrieving and ordering all documents that have the search phrase in them.

The returned search results are then displayed to the doctor via the Doctor Interface Module. The doctors then select the appropriate document from which they wish to extract the information, and this is passed on to the Conversion Module. The Conversion Module then returns a text file to the Data Interface Module for further processing as can be seen in figure 4. The Data Interface Model then uses the Insert Model to determine where the headings are within the text document and uses a marker to demarcate where the headings were found. This marked-up document is then passed on to the Classification Module where marked headings are searched for the keywords relevant to classification. When a particular keyword is found in a heading, the text following that heading up to the next heading occurring in the document is then assigned to the associated attribute matching the keyword.

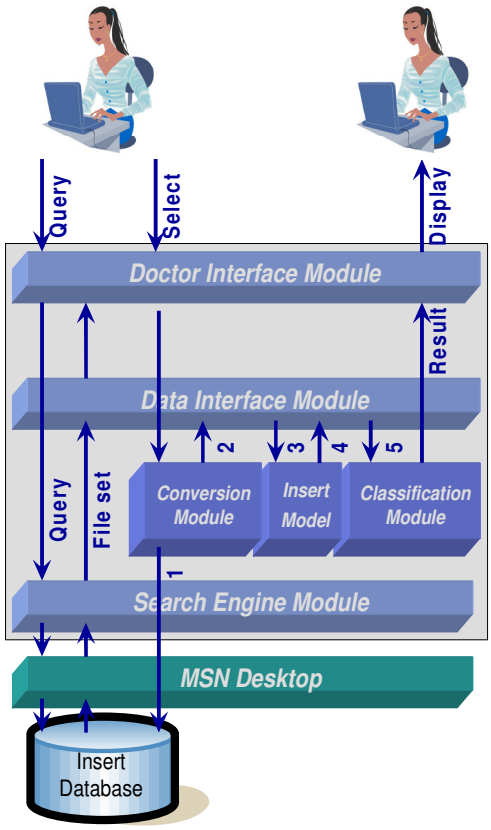

Figure 4. Classification Extraction Inter-Module transaction exchange

When the whole document has been processed, usually most of the attributes will have been ascertained. These are then passed on to the Doctor Interface Module for display.

The ability to extract relevant information can be pushed even further through the application of another insert model. Typically when a doctor wishes to see a generic for a particular medicine, the components of the medication and their associated weights would be submitted to the search engine. Results are then returned from the search engine module for all the files that contain the required search

phrase. However, the existence of such a search phrase in a document may not indicate that it is **feasible** generic. Hence, it becomes necessary to extract the composition of every result that is given from the search engine, using the classification extraction process and ensuring that the search phrase is found within the composition of the medicine related to the document returned as seen in figure 5. When this is confirmed, the file is then submitted as a valid result which can then be selected by the doctor. This process is repeated with every document returned by the initial search until a full list of possible generics is provided. By checking for the original search phrase, the composition component and its weighting in the composition, it becomes far more likely to find a viable generic.

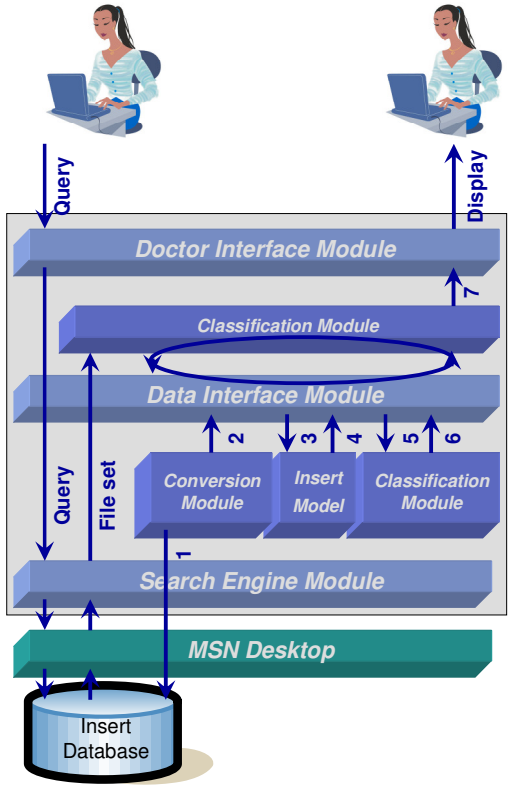

Figure 5. Generic Extraction Inter-Module transaction exchange

VI. RESULTS OF THE STUDY

A. *Meeting the objectives*

At the outset of the study, the medical field was found to be a dynamic and constantly growing field. Being a field where knowledge is inherently descriptive and thus more suited to being contained in textbases, traditional data storage methods are not suitable. Hence it became necessary to show that medical knowledge, in its natural form as part of a textbase, could be usefully be presented through meeting the following aims:

- retrieve relevant documentation in response to a doctor's query,
- extract relevant information quickly and efficiently as per doctor's request so as to aid diagnosis,
- cater for the growing knowledge by easily interacting with newly added documents.

It was postulated that by using powerful search engines to sift through words and retrieve relevant results, **a better supplementation of the doctor's knowledge would result. This postulate was shown to be supported for documentation which was in a descriptive format and which required processing using data-interaction models to enable the relevant knowledge to quickly be extracted. This was supported through the use of MSN Desktop - which presented relevant documentation in response to a doctor's query from which such knowledge could be extracted. Furthermore, through Desktop search any new documentation is automatically indexed and thus easily interfaced to the system.** In doing so all of the primary objectives of the study were attained.

The development and testing phases of the project demonstrated that there was success in interfacing to data. This was done by entering an empty search phrase so that all documentation in the database was retrieved. A number of the documents were selected with a significant success rate of information-extraction being achieved; this was in line with the original 80% success rate requirement set at the beginning. The information was then categorically displayed as seen in figure 6.

B. *Validating data interaction models*

To properly validate the success of the system however, it was necessary to find a sample of users that were willing to use and grade the system on the following criteria:

- Meets a genuine need
- Relevance of results
- Speed of retrieval
- User friendliness
- Information retrieval effectiveness

A group of ten doctors were thus selected for a first pass assessment. These doctors included a group of three General Practitioners and the remainder were specialists from a variety of disciplines. The prototype was then demonstrated, explained and handed over to each doctor for further testing. Upon completion they were required to fill a questionnaire such that each of the above criteria could be measured. An average was then taken of all the doctors' responses to determine the overall success of the prototype. Given the results seen in figure 7, the concept of user friendly data interaction as manifested in the resultant prototype was deemed to be a success.

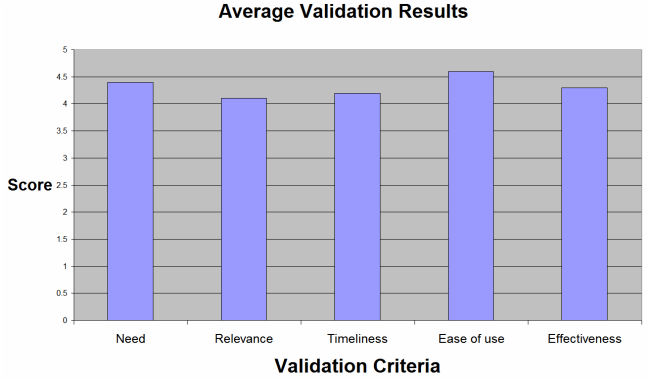

Figure 7. Average Validation Criteria Scores

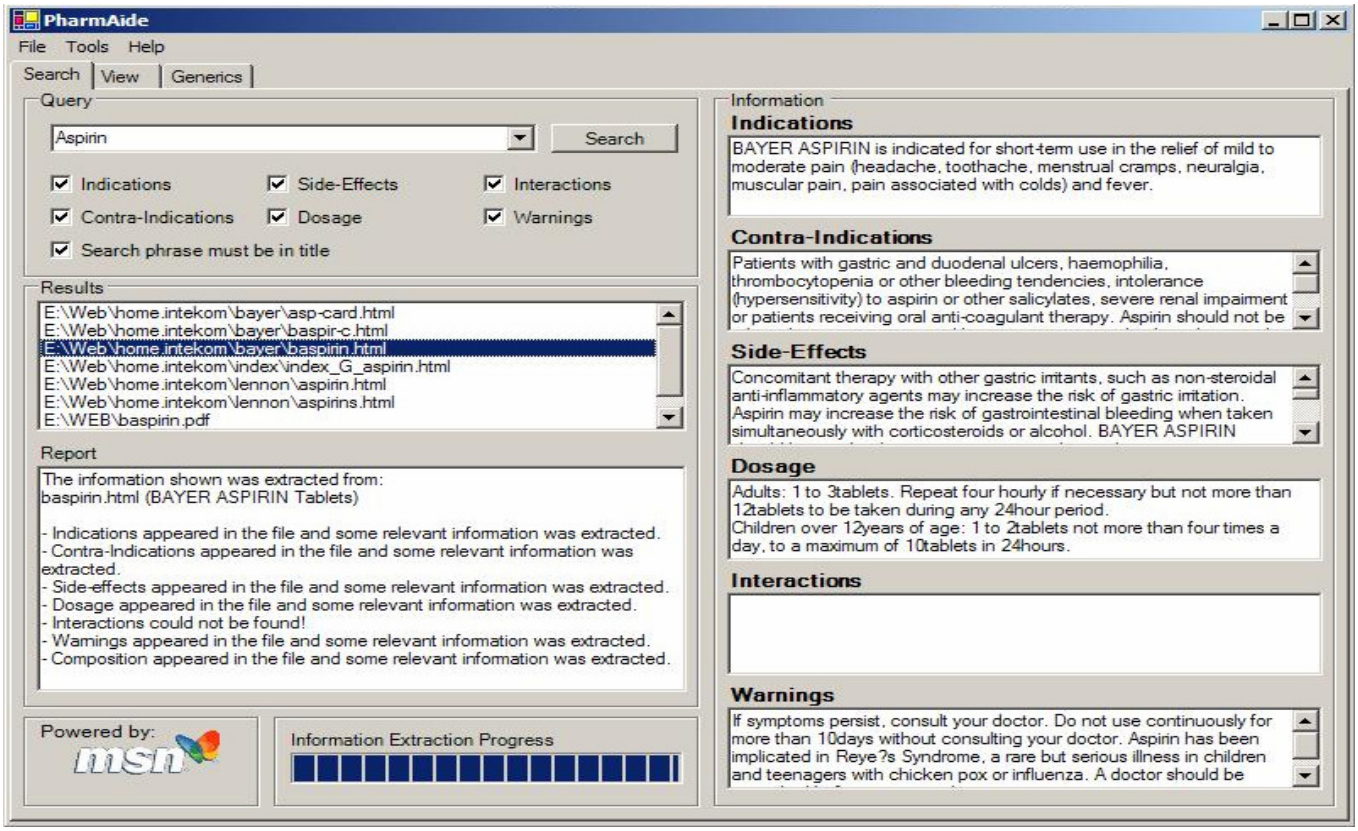

Figure 6. Search page designed to return relevant information in the various documents in which it exists as per user request. Information extracted from [17].

C. *Future replication in descriptive fields*

The purpose of this study went beyond proving the necessity of using searching of textbases to supplement a doctor's knowledge. It determined and then demonstrated a method of analysing a textbase such that desired information could be retrieved. Knowledge is inherently descriptive in a number of fields including medicine. Hence, it is believed that the methods described in this paper could be applied to other fields such that user-friendly data interaction is obtained in other fields. Typically one would take the following steps to replicate the method in other areas:

- Extract a high level overview of the system.
- Identify all the actors in the system and determine the requirements they place on the system.
- Model the actors in the system in the various forms that they exist.
- Implement a Data-Interaction Model accordingly.

VII. Conclusion

There are a number of different fields in which the knowledge contained therein is descriptive. Traditional methods for storing information such as databases have proved to be inadequate, particularly in the medical field, for the given reasons. This research showed that in such cases it becomes necessary to make a *word* the unit of a search. This was achieved through the use of a desktop search engine interacting with a Data-Interaction Model. It was demonstrated that by modelling the actors within a particular system and determining the goals that the primary actor would have in retrieving information, an entirely different, and hence more successful manner of user-data interaction could be constructed.

In doing so, the hypotheses postulated at the beginning of the study were shown to be supported:

1. The system was able to interact effectively with the MSN Desktop to provide relevant documentation in response to a doctor's query as seen in figure 6.
2. The system was able to able to successfully incorporate typical user goals of a doctor to provide relevant information quickly and efficiently as validated by the results shown in figure 7.
3. The system was able to cater for new knowledge by interacting with newly added documentation. This was made possible through MSN Desktop which automatically indexes new documentation and thus makes it searchable by the system as seen in figure 4.

Hence, the study was deemed to be a success.

VIII. Recommendations

At the beginning of the research it was determined that at best, only a beta version of the required software would be developed. Given the success achieved in demonstrating the concept it is felt that further study be conducted into refining the modelling process and improving the final product. The addition of a semantic search engine as well as a spell-checker to improve searching functionality would greatly aid the information retrieval process.

References

[1] Annual Report of the Chief Medical Officer 2002, Intrathecal chemotherapy: keeping up the pressure for safe administration, http://www.dh.gov.uk/PublicationsAndStatistics/Publications/AnnualReports/AnnualReportsBrowsableDocument/fs/en?CONTENT_ID=4

094860&MULTIPAGE_ID=4875362&chk=zxCacz, Last accessed 17 February 2006.
[2] Computer Assisted Medical Diagnosis, http://www.med.univ-rennes1.fr/cerf/publi/ADM_index1.html, Last accessed 2 May 2005.
[3] Solving Information problems in the Pharmaceutical Industry, http://www.verity.com/pdf/3rd_party/RP0050_IDC_Pharma.pdf, Last accessed on 1 May 2005.
[4] Search Diseases Database subject index Diseases Database, http://www.diseasesdatabase.com/begin.asp, Last accessed on 1 May 2005.
[5] MedlinePlus Health Information from the National Library of Medicine, http://www.nlm.nih.gov/medlineplus/medlineplus.html, Last accessed on 1 May 2005.
[6] Food and Drug Administration Home Page, http://www.fda.gov/default.htm, Last accessed on 1 May 2005.
[7] Prescription Drugs, Information, Side Effects, Interactions - Drugs.com, http://www.drugs.com/, Last accessed on 1 May 2005.
[8] Full-Text Searching and Database Content: SearchTools Report, http://www.searchtools.com/info/database-search.html, Last accessed on 1 May 2005.
[9] Introduction to Database Search Skills, http://www.lib.gla.ac.uk/-Docs/Guides/searching.html, Last accessed on 1 May 2005.
[10] Howstuffworks "How Internet Search Engines Work", http://computer.howstuffworks.com/search-engine.htm, Last accessed on 1 May 2005.
[11] Search Engine vs. Database Technology, http://www.unh.edu/NIS/Docs/Search/search-vs-db.html, Last accessed on 1 May 2005.
[12] MSN Toolbar suite vs. Google Desktop, http://insidegoogle.blogspot.com/2004/12/msn-toolbar-suite-vs-google-desktop.html, Last accessed on 1 May 2005.
[13] Google Desktop Still Falls Short of MSN Desktop Search, http://www.windowsdevcenter.com/pub/wlg/6630, Last accessed on 1 May 2005.
[14] Broder A, Taxonomy of Web search, http://www.sigir.org/forum/F2002/broder.pdf, Last accessed on 2 May 2005.
[15] Rose D, Levinson D, Understanding User Goals in Web Search, http://www.ra.ethz.ch/CDstore/www2004/docs/1p13.pdf, Last accessed on 1 May 2005.
[16] From Databases to Information Systems – Information Quality ..., http://www.hiqiq.de/publications/IQ2001.pdf, Last accessed on 1 May 2005.
[17] South African Electronic Package Inserts, http://www.intekom.com/pharm/bayer/baspirin.html, Last accessed 15 March 2006.

APPENDIX

A. PREFACE

Due to the restrictions placed on the length of the paper the following appendices were not included, but are added here to give further detail about the inner workings of the system and how that was manifest in the final prototype. These appendices thus provide a better understanding of the prototype for user friendly data interaction.

A number of benefits can be achieved through a modular system where functionality is grouped. These benefits include:

- The ability to allow for functionality to be improved at a later stage by changing the respective module. Since the system implementation is modular, any additional functionality should have a minimal effect on other modules.
- Maintenance and improvement of the functionality of a particular module is simplified as all the relevant code is contained within a single module.

Given the above benefits it was decided to use a modular approach in developing the prototype. As was indicated in section III of the paper, there are five modules that interact to provide the user with data. **Appendix B.1** deals with the process the prototype uses in implementing the Classification Extraction model. A number of messages are exchanged between the modules to process the initial user input so that relevant information can be extracted and displayed. This message exchange is shown in detail in Appendix B.1.

Appendix B.2 deals with the implementation of the Generic Retrieval Model. Although similar to the Classification Extraction Model in many of the transaction exchanges, the new information need necessitates the interaction with the module implemented to simulate that particular model. This transaction exchange is thus shown in detail in Appendix B.2.

As good an approximation as the implemented models may be, it is necessary to ensure that the user can both make requests as close to their requirement as possible and view the corresponding results in an easily understandable manner. For this to be achieved it was necessary to pay particular attention to the design and layout of the user interface module. Many of the key aspects of the three interaction screens are thus shown in Appendix **C** and its subsections.

Appendix C.1 shows some of the key features implemented to ensure that the user can quickly find the relevant results when necessary. **Appendix C.2** shows the view screen, where a selected document can be viewed for any auxiliary information. A search function was implemented on this screen so that the user can find specified text in the document quickly and efficiently. **Appendix C.3** shows the Generic Extraction Interface and some of the key features used to ensure that relevant documentation is retrieved when necessary.

B. DETAILED MODULE INTERACTIONS

B.1 A DETAILED DESCRIPTION OF THE CLASSIFICATION EXTRACTION PROCESS

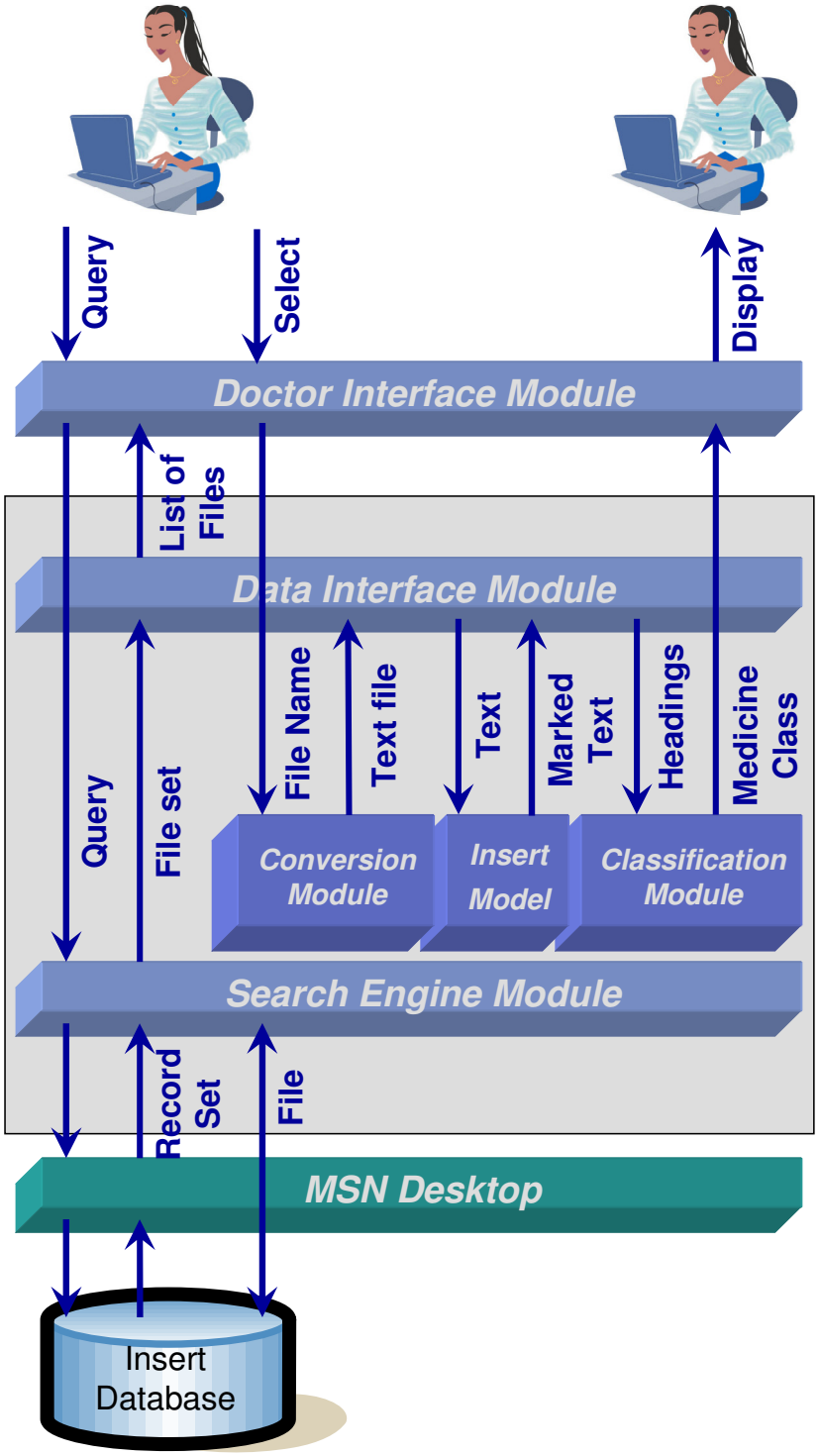

Figure B1: A detailed description of the Classification extraction process

B.2 A DETAILED DESCRIPTION OF THE GENERIC RETRIEVAL PROCESS

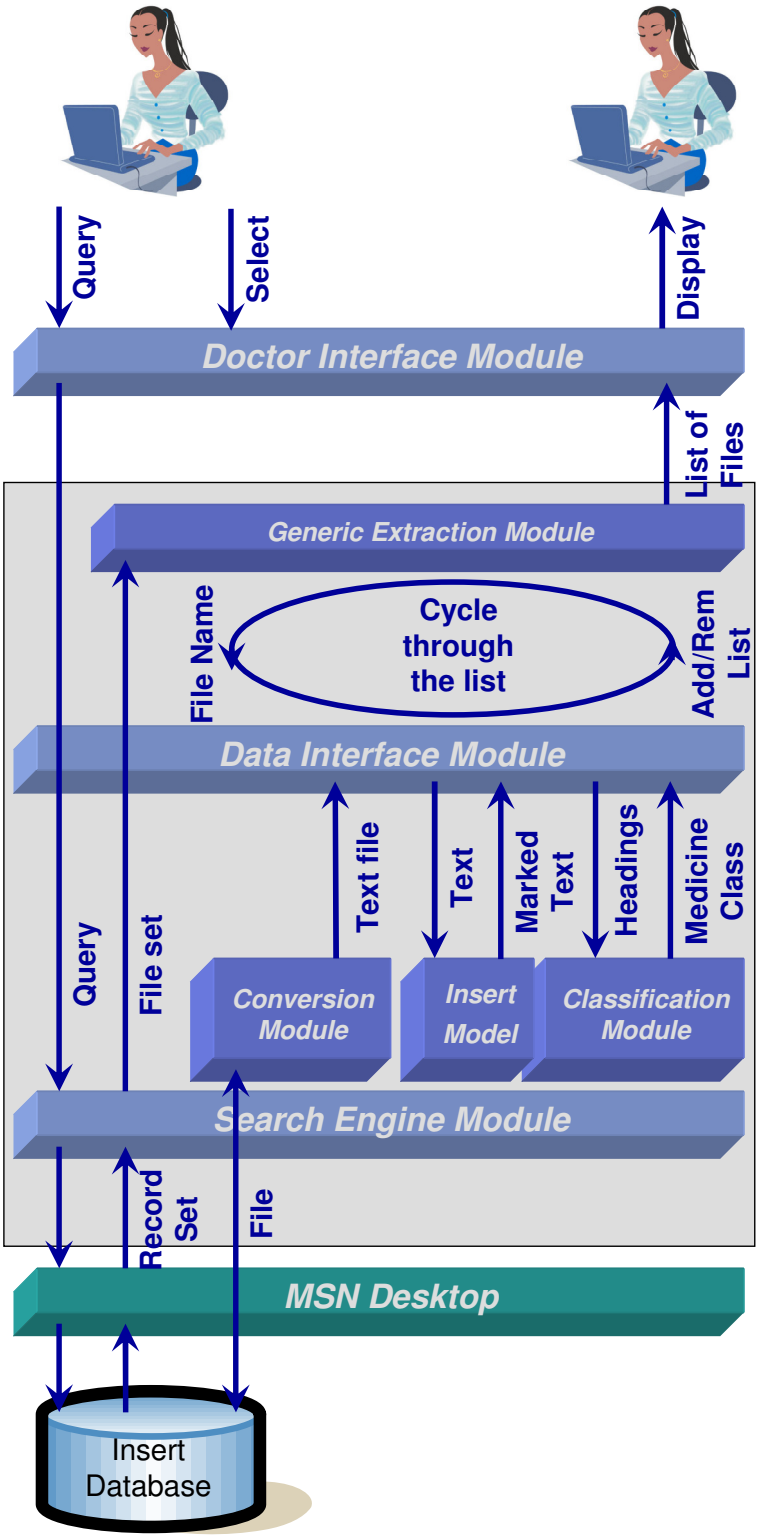

Figure B2: A detailed description of the Generic Retrieval process

C. PROTOTYPE SCREEN-SHOTS

C.1 A DETAILED DESCRIPTION OF THE SEARCH PAGE

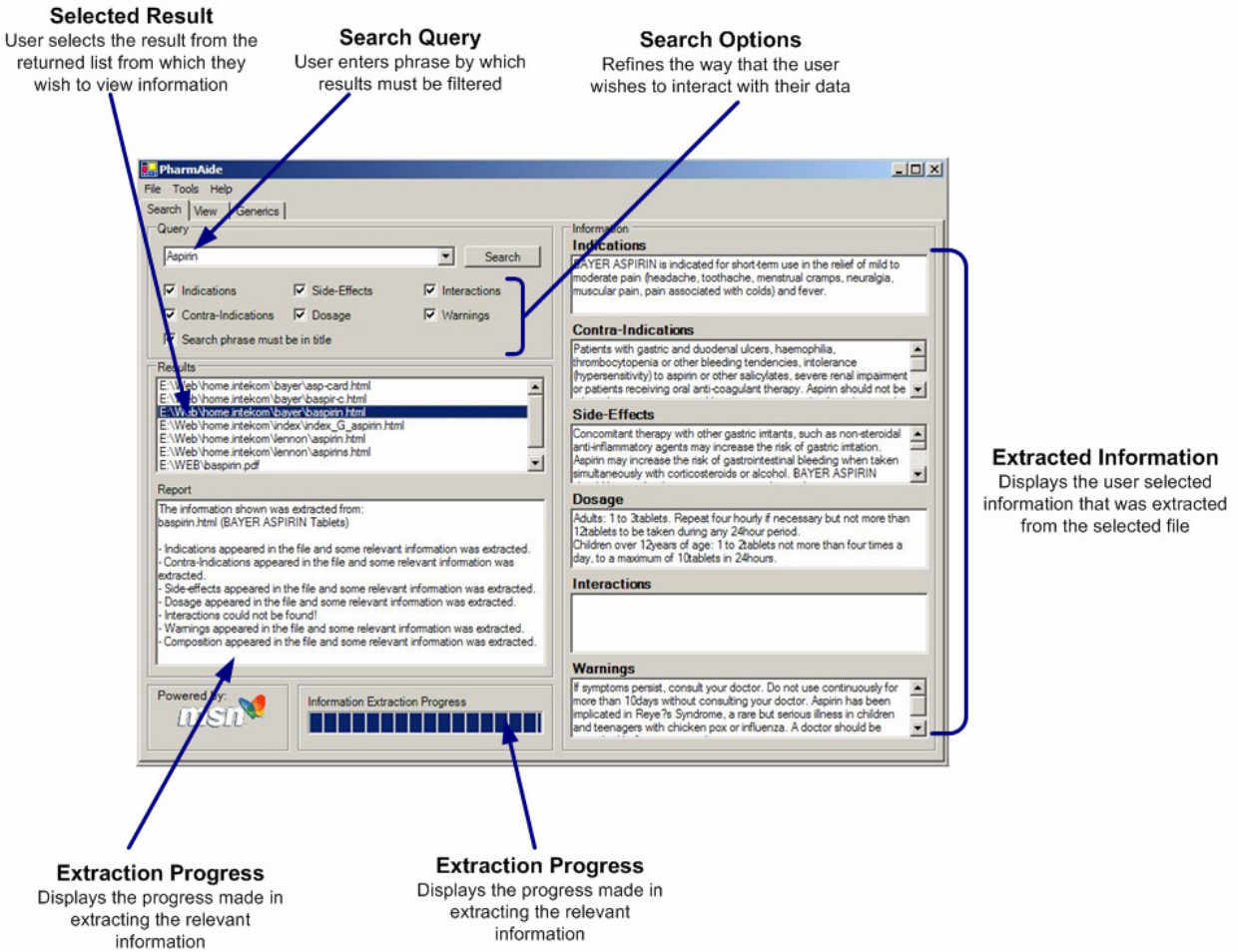

Figure C1: A detailed description of the Search Page

C.2 A DETAILED DESCRIPTION OF THE VIEW PAGE

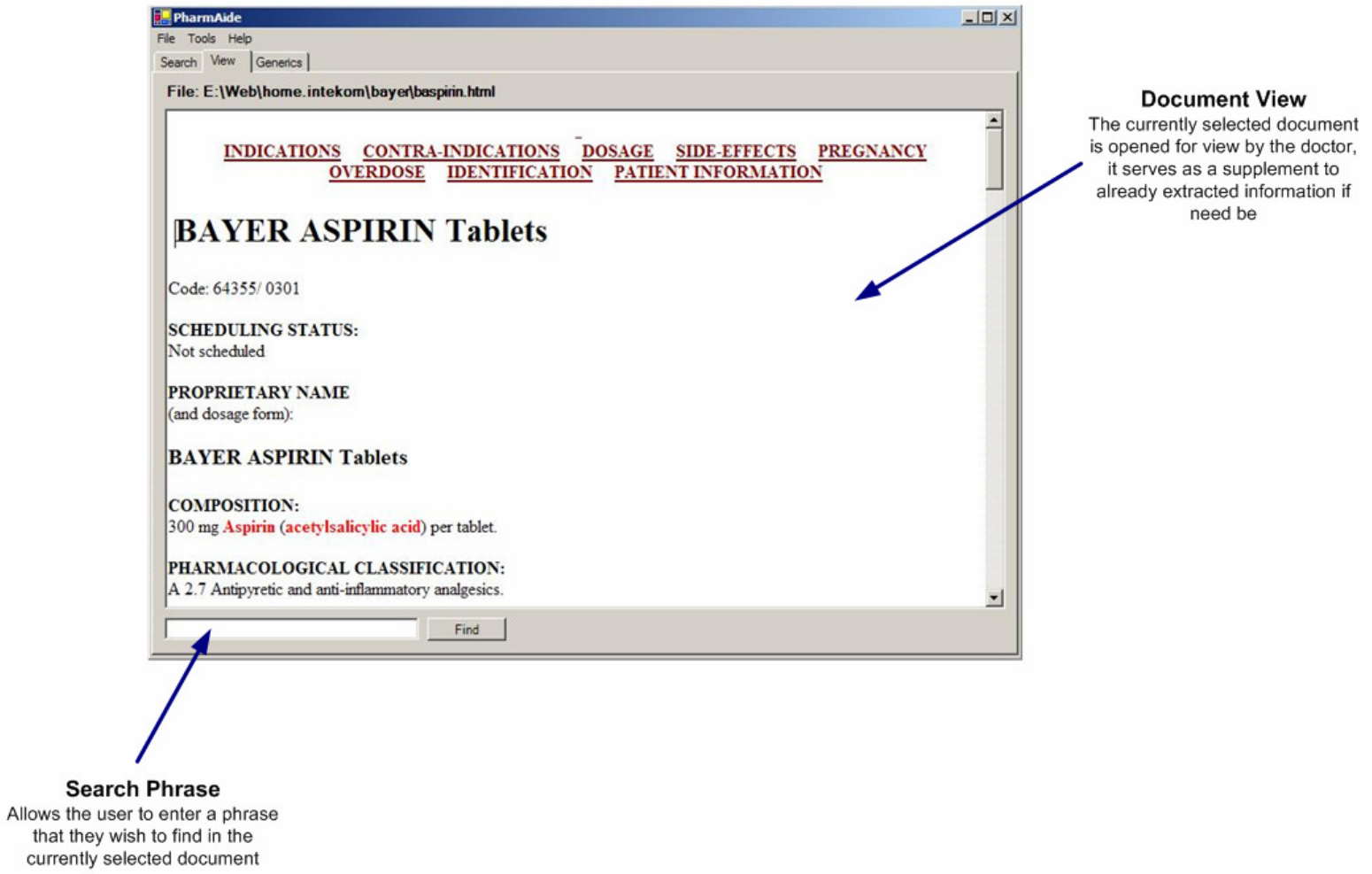

Figure C2: A detailed description of the View Page

C.3 A DETAILED DESCRIPTION OF THE GENERICS PAGE

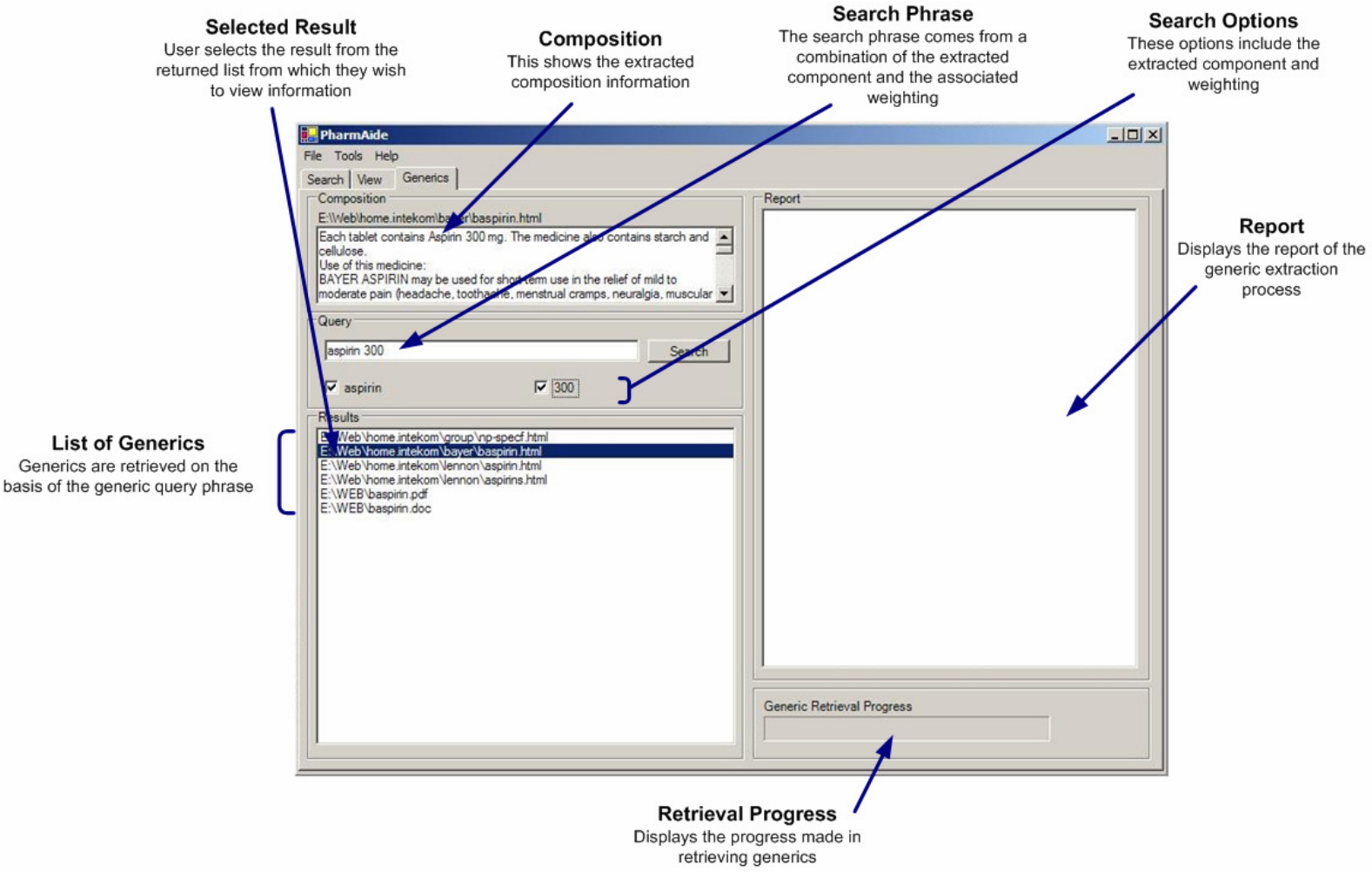

Figure C3: A detailed description of the Generics Page

D. DOCTOR VALIDATION RESULTS

To validate the concept of user friendly data interaction it was necessary to select a group of 10 doctors to score the developed prototype according to the following criteria:

- Meets a genuine need
- Relevance of results
- Speed of retrieval
- User friendliness
- Information retrieval effectiveness

The results are shown in the table below:

Doctor	1	2	3	4	5	6	7	8	9	10	Average
Need	5	5	5	4	4	5	5	4	3	4	4.4
Relevance	5	4	3	3	4	5	5	4	3	5	4.1
Timeliness	4	5	4	4	4	5	4	3	4	5	4.2
Ease of use	4	5	5	5	4	5	5	4	5	4	4.6
Effectiveness	5	4	5	5	3	5	4	4	4	4	4.3

Table D1: A table of the individual doctors' feedback

E. CODE IMPLEMENTATION

This section of the appendix contains code listings to demonstrate the implementation of the design. MSN desktop was used to provide the relevant documentation to the prototype. The MSN Desktop Licensing agreement can be found at http://toolbar.msn.co.za/tou.aspx. A conversion utility by the name of ConvertDoc was also used to perform the conversion function. ConvertDoc is distributed under a free-to-try for non-commercial-use agreement. It is suggested that if this prototype is to be developed further that a new conversion module be written to handle as many different file types as need be.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Threading;
using Microsoft.Windows.DesktopSearch.Query;
using System.Text.RegularExpressions;

namespace PharmAide
{
    /// <summary>
    /// Search is the class primarily responsible for the finding of relevant documents
    and decomposition of the information
    /// </summary>
    public class Search : System.Windows.Forms.Form
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.Windows.Forms.MainMenu mainMenu1;
        private System.Windows.Forms.MenuItem menuItem1;
        private System.Windows.Forms.MenuItem menuItem2;
        private System.Windows.Forms.MenuItem menuItem3;
        private System.Windows.Forms.TabControl tabControl1;
        private System.Windows.Forms.TabPage tabPage1;
        private System.Windows.Forms.TabPage tabPage2;
        private System.Windows.Forms.TabPage tabPage3;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.GroupBox groupBox3;
        private System.Windows.Forms.GroupBox groupBox4;
        private System.Windows.Forms.GroupBox groupBox5;
        private System.Windows.Forms.GroupBox groupBox6;
        private System.Windows.Forms.GroupBox groupBox7;
        private System.Windows.Forms.ListBox listBox1;
        private System.Windows.Forms.ListBox listBox2;
        private System.Windows.Forms.CheckBox checkBox1;
        private System.Windows.Forms.CheckBox checkBox2;
        private System.Windows.Forms.CheckBox checkBox3;
        private System.Windows.Forms.CheckBox checkBox4;
        private System.Windows.Forms.CheckBox checkBox5;
        private System.Windows.Forms.CheckBox checkBox6;
        private System.Windows.Forms.CheckBox checkBox7;
        private System.Windows.Forms.CheckBox checkBox8;
        private System.Windows.Forms.CheckBox checkBox9;
        private System.Windows.Forms.Button FindBTN;
        private System.Windows.Forms.Button SearchBTN;
        private System.Windows.Forms.Button genericQueryBTN;
        private System.Windows.Forms.ComboBox comboBox1;
        private System.Windows.Forms.RichTextBox richTextBox1;
        private System.Windows.Forms.RichTextBox richTextBox2;
        private System.Windows.Forms.RichTextBox richTextBox3;
        private System.Windows.Forms.RichTextBox richTextBox4;
        private System.Windows.Forms.RichTextBox richTextBox5;
        private System.Windows.Forms.RichTextBox richTextBox6;
        private System.Windows.Forms.RichTextBox richTextBox7;
        private System.Windows.Forms.RichTextBox richTextBox8;
        private System.Windows.Forms.RichTextBox richTextBox9;
        private System.Windows.Forms.RichTextBox fileLoadArea;
        private System.Windows.Forms.RichTextBox genericQuery;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.Label label7;
        private System.Windows.Forms.Label label8;
        private System.Windows.Forms.Label compTitle;
        private System.Windows.Forms.MenuItem menuItem5;
        private System.Windows.Forms.MenuItem menuItem6;
```

```
        private System.Windows.Forms.TextBox textBox1;
        private System.ComponentModel.Container components = null;
        private Thread backgroundThread;

        //Below is the list of headings that are of relevance
        private string[] headers = new string[] {"indications", "contra-indications",
    "dosage", "side effects precautions", "composition", "warnings", "interactions"};

        //Define Querybuilder
        private QueryBuilder query;
        //Define FileWatchers
        private FileSystemWatcher textWatcher;
        private FileSystemWatcher rtfWatcher;

        private int checkBoxCounter = 0;
        private int indexFound = 0;
        public int quickSleep = 1000;
        public int slowSleep = 5000;
        private ArrayList Points = new ArrayList();
        private string fileTitle = "";
        private System.Windows.Forms.GroupBox groupBox9;
        private System.Windows.Forms.ProgressBar progressBar1;
        private System.Windows.Forms.GroupBox groupBox8;
        private System.Windows.Forms.Label label9;
        private System.Windows.Forms.PictureBox pictureBox1;
        private System.Windows.Forms.GroupBox groupBox10;
        private System.Windows.Forms.ProgressBar progressBar2;
        private System.Windows.Forms.Label label10;
        private System.Windows.Forms.Label label11;
        private System.Windows.Forms.MenuItem menuItem4;
        private System.Windows.Forms.Button button1;
        private string fileType = "";

        public Search()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();

            //Watch the output files
            textWatcher = new FileSystemWatcher();
            rtfWatcher = new FileSystemWatcher();

            //Instantiate a query builder for performing the search
            this.query = new QueryBuilder();

            //We want some default columns otherwise things don't work
            query.clearColumns();
            query.addColumnList(ColumnType.GeneralColumnList);

            //Same goes for the sort column
            query.sortColumn = ColumnType.GeneralColumns.Rank;

            //Load a list of previously conducted searches
            loadSearchHistory();
        }


        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if(components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }
```

```
        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.Resources.ResourceManager resources = new System.Resources.
    ResourceManager(typeof(Search));
            this.mainMenu1 = new System.Windows.Forms.MainMenu();
            this.menuItem1 = new System.Windows.Forms.MenuItem();
            this.menuItem4 = new System.Windows.Forms.MenuItem();
            this.menuItem5 = new System.Windows.Forms.MenuItem();
            this.menuItem6 = new System.Windows.Forms.MenuItem();
            this.menuItem2 = new System.Windows.Forms.MenuItem();
            this.menuItem3 = new System.Windows.Forms.MenuItem();
            this.tabControl1 = new System.Windows.Forms.TabControl();
            this.tabPage1 = new System.Windows.Forms.TabPage();
            this.groupBox9 = new System.Windows.Forms.GroupBox();
            this.progressBar1 = new System.Windows.Forms.ProgressBar();
            this.label10 = new System.Windows.Forms.Label();
            this.groupBox8 = new System.Windows.Forms.GroupBox();
            this.label9 = new System.Windows.Forms.Label();
            this.pictureBox1 = new System.Windows.Forms.PictureBox();
            this.groupBox3 = new System.Windows.Forms.GroupBox();
            this.listBox1 = new System.Windows.Forms.ListBox();
            this.richTextBox7 = new System.Windows.Forms.RichTextBox();
            this.label8 = new System.Windows.Forms.Label();
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.checkBox7 = new System.Windows.Forms.CheckBox();
            this.SearchBTN = new System.Windows.Forms.Button();
            this.checkBox6 = new System.Windows.Forms.CheckBox();
            this.checkBox4 = new System.Windows.Forms.CheckBox();
            this.checkBox3 = new System.Windows.Forms.CheckBox();
            this.checkBox5 = new System.Windows.Forms.CheckBox();
            this.checkBox2 = new System.Windows.Forms.CheckBox();
            this.checkBox1 = new System.Windows.Forms.CheckBox();
            this.comboBox1 = new System.Windows.Forms.ComboBox();
            this.groupBox2 = new System.Windows.Forms.GroupBox();
            this.richTextBox2 = new System.Windows.Forms.RichTextBox();
            this.richTextBox3 = new System.Windows.Forms.RichTextBox();
            this.richTextBox4 = new System.Windows.Forms.RichTextBox();
            this.richTextBox5 = new System.Windows.Forms.RichTextBox();
            this.richTextBox6 = new System.Windows.Forms.RichTextBox();
            this.richTextBox1 = new System.Windows.Forms.RichTextBox();
            this.label1 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.label6 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.tabPage2 = new System.Windows.Forms.TabPage();
            this.FindBTN = new System.Windows.Forms.Button();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.label7 = new System.Windows.Forms.Label();
            this.fileLoadArea = new System.Windows.Forms.RichTextBox();
            this.tabPage3 = new System.Windows.Forms.TabPage();
            this.groupBox10 = new System.Windows.Forms.GroupBox();
            this.button1 = new System.Windows.Forms.Button();
            this.progressBar2 = new System.Windows.Forms.ProgressBar();
            this.label11 = new System.Windows.Forms.Label();
            this.groupBox5 = new System.Windows.Forms.GroupBox();
            this.richTextBox9 = new System.Windows.Forms.RichTextBox();
            this.groupBox7 = new System.Windows.Forms.GroupBox();
            this.listBox2 = new System.Windows.Forms.ListBox();
            this.groupBox6 = new System.Windows.Forms.GroupBox();
            this.genericQuery = new System.Windows.Forms.RichTextBox();
            this.genericQueryBTN = new System.Windows.Forms.Button();
            this.checkBox9 = new System.Windows.Forms.CheckBox();
            this.checkBox8 = new System.Windows.Forms.CheckBox();
            this.groupBox4 = new System.Windows.Forms.GroupBox();
            this.richTextBox8 = new System.Windows.Forms.RichTextBox();
            this.compTitle = new System.Windows.Forms.Label();
```

```
            this.tabControl1.SuspendLayout();
            this.tabPage1.SuspendLayout();
            this.groupBox9.SuspendLayout();
            this.groupBox8.SuspendLayout();
            this.groupBox3.SuspendLayout();
            this.groupBox1.SuspendLayout();
            this.groupBox2.SuspendLayout();
            this.tabPage2.SuspendLayout();
            this.tabPage3.SuspendLayout();
            this.groupBox10.SuspendLayout();
            this.groupBox5.SuspendLayout();
            this.groupBox7.SuspendLayout();
            this.groupBox6.SuspendLayout();
            this.groupBox4.SuspendLayout();
            this.SuspendLayout();
            //
            // mainMenu1
            //
            this.mainMenu1.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
this.menuItem1,
this.menuItem5,
this.menuItem2});
            //
            // menuItem1
            //
            this.menuItem1.Index = 0;
            this.menuItem1.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
this.menuItem4});
            this.menuItem1.Text = "File";
            //
            // menuItem4
            //
            this.menuItem4.Index = 0;
            this.menuItem4.Text = "Exit";
            this.menuItem4.Click += new System.EventHandler(this.menuItem4_Click);
            //
            // menuItem5
            //
            this.menuItem5.Index = 1;
            this.menuItem5.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
this.menuItem6});
            this.menuItem5.Text = "Tools";
            //
            // menuItem6
            //
            this.menuItem6.Index = 0;
            this.menuItem6.Text = "Settings";
            //
            // menuItem2
            //
            this.menuItem2.Index = 2;
            this.menuItem2.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
this.menuItem3});
            this.menuItem2.Text = "Help";
            //
            // menuItem3
            //
            this.menuItem3.Index = 0;
            this.menuItem3.Text = "About";
            //
            // tabControl1
            //
            this.tabControl1.Controls.Add(this.tabPage1);
            this.tabControl1.Controls.Add(this.tabPage2);
            this.tabControl1.Controls.Add(this.tabPage3);
            this.tabControl1.ItemSize = new System.Drawing.Size(0, 18);
            this.tabControl1.Location = new System.Drawing.Point(0, 0);
            this.tabControl1.Name = "tabControl1";
            this.tabControl1.SelectedIndex = 0;
            this.tabControl1.Size = new System.Drawing.Size(792, 552);
            this.tabControl1.TabIndex = 15;
            //
            // tabPage1
            //
```

```
            this.tabPage1.Controls.Add(this.groupBox9);
            this.tabPage1.Controls.Add(this.groupBox8);
            this.tabPage1.Controls.Add(this.groupBox3);
            this.tabPage1.Controls.Add(this.groupBox1);
            this.tabPage1.Controls.Add(this.groupBox2);
            this.tabPage1.Location = new System.Drawing.Point(4, 22);
            this.tabPage1.Name = "tabPage1";
            this.tabPage1.Size = new System.Drawing.Size(784, 526);
            this.tabPage1.TabIndex = 0;
            this.tabPage1.Text = "Search";
            //
            // groupBox9
            //
            this.groupBox9.Controls.Add(this.progressBar1);
            this.groupBox9.Controls.Add(this.label10);
            this.groupBox9.FlatStyle = System.Windows.Forms.FlatStyle.System;
            this.groupBox9.Location = new System.Drawing.Point(152, 448);
            this.groupBox9.Name = "groupBox9";
            this.groupBox9.Size = new System.Drawing.Size(252, 72);
            this.groupBox9.TabIndex = 28;
            this.groupBox9.TabStop = false;
            //
            // progressBar1
            //
            this.progressBar1.Location = new System.Drawing.Point(10, 32);
            this.progressBar1.Name = "progressBar1";
            this.progressBar1.Size = new System.Drawing.Size(232, 23);
            this.progressBar1.TabIndex = 27;
            //
            // label10
            //
            this.label10.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
    System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)
    (0)));
            this.label10.Location = new System.Drawing.Point(8, 16);
            this.label10.Name = "label10";
            this.label10.Size = new System.Drawing.Size(168, 23);
            this.label10.TabIndex = 28;
            this.label10.Text = "Information Extraction Progress";
            //
            // groupBox8
            //
            this.groupBox8.Controls.Add(this.label9);
            this.groupBox8.Controls.Add(this.pictureBox1);
            this.groupBox8.FlatStyle = System.Windows.Forms.FlatStyle.System;
            this.groupBox8.Location = new System.Drawing.Point(4, 448);
            this.groupBox8.Name = "groupBox8";
            this.groupBox8.Size = new System.Drawing.Size(140, 72);
            this.groupBox8.TabIndex = 27;
            this.groupBox8.TabStop = false;
            //
            // label9
            //
            this.label9.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F, System.
    Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label9.Location = new System.Drawing.Point(8, 8);
            this.label9.Name = "label9";
            this.label9.Size = new System.Drawing.Size(80, 16);
            this.label9.TabIndex = 27;
            this.label9.Text = "Powered by:";
            //
            // pictureBox1
            //
            this.pictureBox1.Image = ((System.Drawing.Image)(resources.GetObject(
    "pictureBox1.Image")));
            this.pictureBox1.Location = new System.Drawing.Point(8, 16);
            this.pictureBox1.Name = "pictureBox1";
            this.pictureBox1.Size = new System.Drawing.Size(128, 50);
            this.pictureBox1.TabIndex = 26;
            this.pictureBox1.TabStop = false;
            //
            // groupBox3
            //
            this.groupBox3.Controls.Add(this.listBox1);
```

```
            this.groupBox3.Controls.Add(this.richTextBox7);
            this.groupBox3.Controls.Add(this.label8);
            this.groupBox3.Location = new System.Drawing.Point(4, 136);
            this.groupBox3.Name = "groupBox3";
            this.groupBox3.Size = new System.Drawing.Size(400, 312);
            this.groupBox3.TabIndex = 16;
            this.groupBox3.TabStop = false;
            this.groupBox3.Text = "Results";
            //
            // listBox1
            //
            this.listBox1.Location = new System.Drawing.Point(8, 16);
            this.listBox1.Name = "listBox1";
            this.listBox1.Size = new System.Drawing.Size(384, 95);
            this.listBox1.TabIndex = 0;
            this.listBox1.SelectedIndexChanged += new System.EventHandler(this.
    listBox1_SelectedIndexChanged);
            //
            // richTextBox7
            //
            this.richTextBox7.Location = new System.Drawing.Point(8, 136);
            this.richTextBox7.Name = "richTextBox7";
            this.richTextBox7.Size = new System.Drawing.Size(384, 164);
            this.richTextBox7.TabIndex = 22;
            this.richTextBox7.Text = "";
            //
            // label8
            //
            this.label8.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
    System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)
    (0)));
            this.label8.Location = new System.Drawing.Point(8, 120);
            this.label8.Name = "label8";
            this.label8.TabIndex = 22;
            this.label8.Text = "Report";
            //
            // groupBox1
            //
            this.groupBox1.Controls.Add(this.checkBox7);
            this.groupBox1.Controls.Add(this.SearchBTN);
            this.groupBox1.Controls.Add(this.checkBox6);
            this.groupBox1.Controls.Add(this.checkBox4);
            this.groupBox1.Controls.Add(this.checkBox3);
            this.groupBox1.Controls.Add(this.checkBox5);
            this.groupBox1.Controls.Add(this.checkBox2);
            this.groupBox1.Controls.Add(this.checkBox1);
            this.groupBox1.Controls.Add(this.comboBox1);
            this.groupBox1.Location = new System.Drawing.Point(4, 0);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(400, 136);
            this.groupBox1.TabIndex = 15;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Query";
            //
            // checkBox7
            //
            this.checkBox7.Location = new System.Drawing.Point(16, 104);
            this.checkBox7.Name = "checkBox7";
            this.checkBox7.Size = new System.Drawing.Size(176, 24);
            this.checkBox7.TabIndex = 18;
            this.checkBox7.Text = "Search phrase must be in title";
            //
            // SearchBTN
            //
            this.SearchBTN.Location = new System.Drawing.Point(312, 24);
            this.SearchBTN.Name = "SearchBTN";
            this.SearchBTN.Size = new System.Drawing.Size(75, 21);
            this.SearchBTN.TabIndex = 17;
            this.SearchBTN.Text = "Search";
            this.SearchBTN.Click += new System.EventHandler(this.SearchBTN_Click);
            //
            // checkBox6
            //
            this.checkBox6.Location = new System.Drawing.Point(272, 80);
```

```
            this.checkBox6.Name = "checkBox6";
            this.checkBox6.TabIndex = 16;
            this.checkBox6.Text = "Warnings";
            this.checkBox6.CheckedChanged += new System.EventHandler(this.
    checkBox6_CheckedChanged);
            //
            // checkBox4
            //
            this.checkBox4.Location = new System.Drawing.Point(272, 56);
            this.checkBox4.Name = "checkBox4";
            this.checkBox4.TabIndex = 15;
            this.checkBox4.Text = "Interactions";
            this.checkBox4.CheckedChanged += new System.EventHandler(this.
    checkBox4_CheckedChanged);
            //
            // checkBox3
            //
            this.checkBox3.Location = new System.Drawing.Point(144, 80);
            this.checkBox3.Name = "checkBox3";
            this.checkBox3.TabIndex = 14;
            this.checkBox3.Text = "Dosage";
            this.checkBox3.CheckedChanged += new System.EventHandler(this.
    checkBox3_CheckedChanged);
            //
            // checkBox5
            //
            this.checkBox5.Location = new System.Drawing.Point(144, 56);
            this.checkBox5.Name = "checkBox5";
            this.checkBox5.TabIndex = 13;
            this.checkBox5.Text = "Side-Effects";
            this.checkBox5.CheckedChanged += new System.EventHandler(this.
    checkBox5_CheckedChanged);
            //
            // checkBox2
            //
            this.checkBox2.Location = new System.Drawing.Point(16, 80);
            this.checkBox2.Name = "checkBox2";
            this.checkBox2.Size = new System.Drawing.Size(120, 24);
            this.checkBox2.TabIndex = 12;
            this.checkBox2.Text = "Contra-Indications";
            this.checkBox2.CheckedChanged += new System.EventHandler(this.
    checkBox2_CheckedChanged);
            //
            // checkBox1
            //
            this.checkBox1.Location = new System.Drawing.Point(16, 56);
            this.checkBox1.Name = "checkBox1";
            this.checkBox1.TabIndex = 11;
            this.checkBox1.Text = "Indications";
            this.checkBox1.CheckedChanged += new System.EventHandler(this.
    checkBox1_CheckedChanged);
            //
            // comboBox1
            //
            this.comboBox1.Location = new System.Drawing.Point(16, 24);
            this.comboBox1.Name = "comboBox1";
            this.comboBox1.Size = new System.Drawing.Size(288, 21);
            this.comboBox1.TabIndex = 0;
            //
            // groupBox2
            //
            this.groupBox2.Controls.Add(this.richTextBox2);
            this.groupBox2.Controls.Add(this.richTextBox3);
            this.groupBox2.Controls.Add(this.richTextBox4);
            this.groupBox2.Controls.Add(this.richTextBox5);
            this.groupBox2.Controls.Add(this.richTextBox6);
            this.groupBox2.Controls.Add(this.richTextBox1);
            this.groupBox2.Controls.Add(this.label1);
            this.groupBox2.Controls.Add(this.label3);
            this.groupBox2.Controls.Add(this.label4);
            this.groupBox2.Controls.Add(this.label5);
            this.groupBox2.Controls.Add(this.label6);
            this.groupBox2.Controls.Add(this.label2);
            this.groupBox2.Location = new System.Drawing.Point(412, 0);
```

```
            this.groupBox2.Name = "groupBox2";
            this.groupBox2.Size = new System.Drawing.Size(368, 520);
            this.groupBox2.TabIndex = 17;
            this.groupBox2.TabStop = false;
            this.groupBox2.Text = "Information";
            //
            // richTextBox2
            //
            this.richTextBox2.Location = new System.Drawing.Point(8, 32);
            this.richTextBox2.Name = "richTextBox2";
            this.richTextBox2.Size = new System.Drawing.Size(352, 60);
            this.richTextBox2.TabIndex = 1;
            this.richTextBox2.Text = "";
            this.richTextBox2.Visible = false;
            //
            // richTextBox3
            //
            this.richTextBox3.Location = new System.Drawing.Point(8, 198);
            this.richTextBox3.Name = "richTextBox3";
            this.richTextBox3.Size = new System.Drawing.Size(352, 60);
            this.richTextBox3.TabIndex = 3;
            this.richTextBox3.Text = "";
            this.richTextBox3.Visible = false;
            //
            // richTextBox4
            //
            this.richTextBox4.Location = new System.Drawing.Point(8, 281);
            this.richTextBox4.Name = "richTextBox4";
            this.richTextBox4.Size = new System.Drawing.Size(352, 60);
            this.richTextBox4.TabIndex = 4;
            this.richTextBox4.Text = "";
            this.richTextBox4.Visible = false;
            //
            // richTextBox5
            //
            this.richTextBox5.Location = new System.Drawing.Point(8, 364);
            this.richTextBox5.Name = "richTextBox5";
            this.richTextBox5.Size = new System.Drawing.Size(352, 60);
            this.richTextBox5.TabIndex = 5;
            this.richTextBox5.Text = "";
            this.richTextBox5.Visible = false;
            //
            // richTextBox6
            //
            this.richTextBox6.Location = new System.Drawing.Point(8, 447);
            this.richTextBox6.Name = "richTextBox6";
            this.richTextBox6.Size = new System.Drawing.Size(352, 60);
            this.richTextBox6.TabIndex = 14;
            this.richTextBox6.Text = "";
            this.richTextBox6.Visible = false;
            //
            // richTextBox1
            //
            this.richTextBox1.Location = new System.Drawing.Point(8, 115);
            this.richTextBox1.Name = "richTextBox1";
            this.richTextBox1.Size = new System.Drawing.Size(352, 60);
            this.richTextBox1.TabIndex = 2;
            this.richTextBox1.Text = "";
            this.richTextBox1.Visible = false;
            //
            // label1
            //
            this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System↙
    .Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label1.Location = new System.Drawing.Point(8, 15);
            this.label1.Name = "label1";
            this.label1.TabIndex = 16;
            this.label1.Text = "Indications";
            this.label1.Visible = false;
            //
            // label3
            //
            this.label3.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System↙
    .Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
```

```
            this.label3.Location = new System.Drawing.Point(8, 181);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(128, 23);
            this.label3.TabIndex = 18;
            this.label3.Text = "Side-Effects";
            this.label3.Visible = false;
            //
            // label4
            //
            this.label4.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System↙
    .Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label4.Location = new System.Drawing.Point(8, 264);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(128, 23);
            this.label4.TabIndex = 19;
            this.label4.Text = "Dosage";
            this.label4.Visible = false;
            //
            // label5
            //
            this.label5.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System↙
    .Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label5.Location = new System.Drawing.Point(8, 347);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(128, 23);
            this.label5.TabIndex = 20;
            this.label5.Text = "Interactions";
            this.label5.Visible = false;
            //
            // label6
            //
            this.label6.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System↙
    .Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label6.Location = new System.Drawing.Point(8, 430);
            this.label6.Name = "label6";
            this.label6.Size = new System.Drawing.Size(128, 23);
            this.label6.TabIndex = 21;
            this.label6.Text = "Warnings";
            this.label6.Visible = false;
            //
            // label2
            //
            this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System↙
    .Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label2.Location = new System.Drawing.Point(8, 98);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(128, 23);
            this.label2.TabIndex = 17;
            this.label2.Text = "Contra-Indications";
            this.label2.Visible = false;
            //
            // tabPage2
            //
            this.tabPage2.Controls.Add(this.FindBTN);
            this.tabPage2.Controls.Add(this.textBox1);
            this.tabPage2.Controls.Add(this.label7);
            this.tabPage2.Controls.Add(this.fileLoadArea);
            this.tabPage2.Location = new System.Drawing.Point(4, 22);
            this.tabPage2.Name = "tabPage2";
            this.tabPage2.Size = new System.Drawing.Size(784, 526);
            this.tabPage2.TabIndex = 1;
            this.tabPage2.Text = "View";
            //
            // FindBTN
            //
            this.FindBTN.Location = new System.Drawing.Point(248, 496);
            this.FindBTN.Name = "FindBTN";
            this.FindBTN.Size = new System.Drawing.Size(72, 21);
            this.FindBTN.TabIndex = 20;
            this.FindBTN.Text = "Find";
            this.FindBTN.Click += new System.EventHandler(this.FindBTN_Click);
            //
            // textBox1
            //
```

```
            this.textBox1.Location = new System.Drawing.Point(8, 496);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(232, 20);
            this.textBox1.TabIndex = 19;
            this.textBox1.Text = "";
            this.textBox1.TextChanged += new System.EventHandler(this.
    textBox1_TextChanged);
            //
            // label7
            //
            this.label7.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System
    .Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label7.Location = new System.Drawing.Point(8, 8);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(768, 23);
            this.label7.TabIndex = 18;
            this.label7.Text = "File Contents";
            //
            // fileLoadArea
            //
            this.fileLoadArea.Font = new System.Drawing.Font("Arial", 12F, System.Drawing.
    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.fileLoadArea.Location = new System.Drawing.Point(8, 32);
            this.fileLoadArea.Name = "fileLoadArea";
            this.fileLoadArea.Size = new System.Drawing.Size(768, 456);
            this.fileLoadArea.TabIndex = 2;
            this.fileLoadArea.Text = "";
            //
            // tabPage3
            //
            this.tabPage3.Controls.Add(this.groupBox10);
            this.tabPage3.Controls.Add(this.groupBox5);
            this.tabPage3.Controls.Add(this.groupBox7);
            this.tabPage3.Controls.Add(this.groupBox6);
            this.tabPage3.Controls.Add(this.groupBox4);
            this.tabPage3.Location = new System.Drawing.Point(4, 22);
            this.tabPage3.Name = "tabPage3";
            this.tabPage3.Size = new System.Drawing.Size(784, 526);
            this.tabPage3.TabIndex = 2;
            this.tabPage3.Text = "Generics";
            //
            // groupBox10
            //
            this.groupBox10.Controls.Add(this.button1);
            this.groupBox10.Controls.Add(this.progressBar2);
            this.groupBox10.Controls.Add(this.label11);
            this.groupBox10.FlatStyle = System.Windows.Forms.FlatStyle.System;
            this.groupBox10.Location = new System.Drawing.Point(412, 448);
            this.groupBox10.Name = "groupBox10";
            this.groupBox10.Size = new System.Drawing.Size(364, 72);
            this.groupBox10.TabIndex = 29;
            this.groupBox10.TabStop = false;
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(280, 32);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(72, 24);
            this.button1.TabIndex = 30;
            this.button1.Text = "Cancel";
            this.button1.Visible = false;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // progressBar2
            //
            this.progressBar2.Location = new System.Drawing.Point(10, 32);
            this.progressBar2.Name = "progressBar2";
            this.progressBar2.Size = new System.Drawing.Size(262, 23);
            this.progressBar2.TabIndex = 27;
            //
            // label11
            //
            this.label11.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
    System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)
```

```
(0)));
        this.label11.Location = new System.Drawing.Point(8, 16);
        this.label11.Name = "label11";
        this.label11.Size = new System.Drawing.Size(168, 23);
        this.label11.TabIndex = 29;
        this.label11.Text = "Generic Retrieval Progress";
        //
        // groupBox5
        //
        this.groupBox5.Controls.Add(this.richTextBox9);
        this.groupBox5.Location = new System.Drawing.Point(412, 0);
        this.groupBox5.Name = "groupBox5";
        this.groupBox5.Size = new System.Drawing.Size(364, 448);
        this.groupBox5.TabIndex = 23;
        this.groupBox5.TabStop = false;
        this.groupBox5.Text = "Report";
        //
        // richTextBox9
        //
        this.richTextBox9.Location = new System.Drawing.Point(8, 16);
        this.richTextBox9.Name = "richTextBox9";
        this.richTextBox9.Size = new System.Drawing.Size(344, 420);
        this.richTextBox9.TabIndex = 22;
        this.richTextBox9.Text = "";
        //
        // groupBox7
        //
        this.groupBox7.Controls.Add(this.listBox2);
        this.groupBox7.Location = new System.Drawing.Point(4, 200);
        this.groupBox7.Name = "groupBox7";
        this.groupBox7.Size = new System.Drawing.Size(400, 320);
        this.groupBox7.TabIndex = 22;
        this.groupBox7.TabStop = false;
        this.groupBox7.Text = "Results";
        //
        // listBox2
        //
        this.listBox2.Location = new System.Drawing.Point(8, 16);
        this.listBox2.Name = "listBox2";
        this.listBox2.Size = new System.Drawing.Size(384, 290);
        this.listBox2.TabIndex = 0;
        this.listBox2.SelectedIndexChanged += new System.EventHandler(this.
listBox2_SelectedIndexChanged);
        //
        // groupBox6
        //
        this.groupBox6.Controls.Add(this.genericQuery);
        this.groupBox6.Controls.Add(this.genericQueryBTN);
        this.groupBox6.Controls.Add(this.checkBox9);
        this.groupBox6.Controls.Add(this.checkBox8);
        this.groupBox6.Location = new System.Drawing.Point(4, 112);
        this.groupBox6.Name = "groupBox6";
        this.groupBox6.Size = new System.Drawing.Size(400, 88);
        this.groupBox6.TabIndex = 21;
        this.groupBox6.TabStop = false;
        this.groupBox6.Text = "Query";
        //
        // genericQuery
        //
        this.genericQuery.Location = new System.Drawing.Point(16, 24);
        this.genericQuery.Name = "genericQuery";
        this.genericQuery.Size = new System.Drawing.Size(288, 20);
        this.genericQuery.TabIndex = 19;
        this.genericQuery.Text = "";
        //
        // genericQueryBTN
        //
        this.genericQueryBTN.Location = new System.Drawing.Point(312, 24);
        this.genericQueryBTN.Name = "genericQueryBTN";
        this.genericQueryBTN.Size = new System.Drawing.Size(75, 21);
        this.genericQueryBTN.TabIndex = 17;
        this.genericQueryBTN.Text = "Search";
        this.genericQueryBTN.Click += new System.EventHandler(this.
genericQueryBtn_Click);
```

```
            //
            // checkBox9
            //
            this.checkBox9.Location = new System.Drawing.Point(208, 56);
            this.checkBox9.Name = "checkBox9";
            this.checkBox9.Size = new System.Drawing.Size(176, 24);
            this.checkBox9.TabIndex = 12;
            this.checkBox9.Text = "Weight1";
            this.checkBox9.Visible = false;
            this.checkBox9.CheckedChanged += new System.EventHandler(this.
    checkBox9_CheckedChanged);
            //
            // checkBox8
            //
            this.checkBox8.Location = new System.Drawing.Point(16, 56);
            this.checkBox8.Name = "checkBox8";
            this.checkBox8.Size = new System.Drawing.Size(176, 24);
            this.checkBox8.TabIndex = 11;
            this.checkBox8.Text = "Ingr1";
            this.checkBox8.Visible = false;
            this.checkBox8.CheckedChanged += new System.EventHandler(this.
    checkBox8_CheckedChanged);
            //
            // groupBox4
            //
            this.groupBox4.Controls.Add(this.richTextBox8);
            this.groupBox4.Controls.Add(this.compTitle);
            this.groupBox4.Location = new System.Drawing.Point(4, 0);
            this.groupBox4.Name = "groupBox4";
            this.groupBox4.Size = new System.Drawing.Size(400, 112);
            this.groupBox4.TabIndex = 19;
            this.groupBox4.TabStop = false;
            this.groupBox4.Text = "Composition";
            //
            // richTextBox8
            //
            this.richTextBox8.Location = new System.Drawing.Point(8, 32);
            this.richTextBox8.Name = "richTextBox8";
            this.richTextBox8.Size = new System.Drawing.Size(384, 72);
            this.richTextBox8.TabIndex = 18;
            this.richTextBox8.Text = "";
            //
            // compTitle
            //
            this.compTitle.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
    System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)
    (0)));
            this.compTitle.Location = new System.Drawing.Point(8, 18);
            this.compTitle.Name = "compTitle";
            this.compTitle.Size = new System.Drawing.Size(384, 23);
            this.compTitle.TabIndex = 24;
            this.compTitle.Text = "Medicine";
            //
            // Search
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.ClientSize = new System.Drawing.Size(792, 553);
            this.Controls.Add(this.tabControl1);
            this.Menu = this.mainMenu1;
            this.Name = "Search";
            this.Text = "PharmAide";
            this.Closing += new System.ComponentModel.CancelEventHandler(this.
    Search_Closing);
            this.tabControl1.ResumeLayout(false);
            this.tabPage1.ResumeLayout(false);
            this.groupBox9.ResumeLayout(false);
            this.groupBox8.ResumeLayout(false);
            this.groupBox3.ResumeLayout(false);
            this.groupBox1.ResumeLayout(false);
            this.groupBox2.ResumeLayout(false);
            this.tabPage2.ResumeLayout(false);
            this.tabPage3.ResumeLayout(false);
            this.groupBox10.ResumeLayout(false);
            this.groupBox5.ResumeLayout(false);
```

```
            this.groupBox7.ResumeLayout(false);
            this.groupBox6.ResumeLayout(false);
            this.groupBox4.ResumeLayout(false);
            this.ResumeLayout(false);

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new Search());
        }


        /// <summary>
        /// Parse the text and fill the Medicine attributes
        /// </summary>
        public int ParseText(string text, ref Medicine customClass)
        {
            //Make sure that the class has been instantiated
            if(customClass == null)
            {
                customClass = new Medicine();
            }

            //Read info from the line
            if(text.IndexOf("|") > -1)  //Make sure this is a line we should read by searching for heading marker '|'
            {
                string[] parts = text.Split('|');
                //Create a string to combine the information related to a specific title
                string info = "";
                for (int i = 1; i < parts.Length; i++)
                    info += parts[i];

                //Try and determine a match for the title
                int match = 0;
                string delimStr = " ,.:;-()";
                string heading = "";
                string headerWord, word;

                //Remove any delimiters to find the words in the heading phrase and see of there are any matches to predefined headings
                char [] delimiter = delimStr.ToCharArray();
                string[] words = parts[0].Trim().ToLower().Split(delimiter);
                string[] headerWords;

                //Cycle through the Predefined headings to see if the phrase matches them
                for (int i = 0; i < 7; i++)
                {
                    match = 0;
                    headerWords = headers[i].ToString().Trim().ToLower().Split(delimiter); //Remove the delimiters
                    for (int j = 0; j < words.Length; j++)
                    {
                        for (int k = 0; k < headerWords.Length; k++)
                        {
                            word = words[j].TrimEnd('s');
                            headerWord = headerWords[k].TrimEnd('s');
                            if(word == headerWord)
                                match ++;
                        }
                    }
                    //If all the words in the predefined headings are found in the heading phrase a match is created
                    if(match == headerWords.Length)
                        heading = headers[i].ToString().Trim().ToLower();
                }

                // Use the matched Heading to assign info to the right medicine attribute
```

```
            if(parts.Length > 1)    //Make sure we have at least two values
            {
                switch(heading)
                {
                    case "indications":
                        customClass.Indications = info;
                        return 1;
                        break;  // In c# you MUST break each case
                    case "contra-indications":
                        customClass.ContraIndications = info;
                        return 1;
                        break;  // In c# you MUST break each case
                    case "dosage":
                        customClass.Dosage = info;
                        return 1;
                        break;  // In c# you MUST break each case
                    case "side effects precautions":
                        customClass.SideEffects = info;
                        return 1;
                        break;  // In c# you MUST break each case
                    case "composition":
                        customClass.Composition = info;
                        return 1;
                        break;  // In c# you MUST break each case
                    case "warnings":
                        customClass.Warnings = info;
                        return 1;
                        break;  // In c# you MUST break each case
                    case "interactions":
                        customClass.Interactions = info;
                        return 1;
                        break;  // In c# you MUST break each case
                }
            }
        }
        return 0;
    }


    /// <summary>
    /// Read the file contents and display it in the View tab
    /// </summary>
    private void ReadFileInfo(String filename)
    {
        try
        {
            FileStream fs = new FileStream(filename, FileMode.Open, FileAccess.Read);
            FileInfo fInfo = new FileInfo(filename);

            //Determine the type of the file before displaying the contents
            string fext = fInfo.Extension.ToUpper();
            if (fext.Equals(".RTF"))
                fileLoadArea.LoadFile(fs, RichTextBoxStreamType.RichText);
            else
                fileLoadArea.LoadFile(fs, RichTextBoxStreamType.PlainText);
            fs.Close(); //Close the File stream when done
        }
        catch(Exception e)
        {
            throw new Exception(e.Message);
        }
    }


    /// <summary>
    /// Determine which information is shown and where and how much is shown
    /// </summary>
    private void PositionResults()
    {
        Points.Clear();
        if(checkBoxCounter > 0)
        {
            //Determine the height of the results box
            richTextBox1.Height = (475 - 23*(checkBoxCounter - 1))/checkBoxCounter;
```

```
            richTextBox2.Height = (475 - 23*(checkBoxCounter - 1))/checkBoxCounter;
            richTextBox3.Height = (475 - 23*(checkBoxCounter - 1))/checkBoxCounter;
            richTextBox4.Height = (475 - 23*(checkBoxCounter - 1))/checkBoxCounter;
            richTextBox5.Height = (475 - 23*(checkBoxCounter - 1))/checkBoxCounter;
            richTextBox6.Height = (475 - 23*(checkBoxCounter - 1))/checkBoxCounter;

            //Determine the pointer positions of the results box
            for (int i = 0; i <= 5; i++)
            {
                Point point;
                point = new Point(8, 32 + 23*i + (475 - 23*(checkBoxCounter - 1))/
checkBoxCounter*i);
                Points.Add(point);
            }

            //Place Checked Boxes onto positions determined
            int j = 0;
            if(checkBox1.CheckState == CheckState.Checked)
            {
                Point point = (Point) Points[j];
                label1.Location = new Point(point.X, point.Y - 17);
                richTextBox2.Location = point;
                j = j + 1;
            }
            if(checkBox2.CheckState == CheckState.Checked)
            {
                Point point = (Point) Points[j];
                label2.Location = new Point(point.X, point.Y - 17);
                richTextBox1.Location = point;
                j = j + 1;
            }
            if(checkBox5.CheckState == CheckState.Checked)
            {
                Point point = (Point) Points[j];
                label3.Location = new Point(point.X, point.Y - 17);
                richTextBox3.Location = point;
                j = j + 1;
            }
            if(checkBox3.CheckState == CheckState.Checked)
            {
                Point point = (Point) Points[j];
                label4.Location = new Point(point.X, point.Y - 17);
                richTextBox4.Location = point;
                j = j + 1;
            }
            if(checkBox4.CheckState == CheckState.Checked)
            {
                Point point = (Point) Points[j];
                label5.Location = new Point(point.X, point.Y - 17);
                richTextBox5.Location = point;
                j = j + 1;
            }
            if(checkBox6.CheckState == CheckState.Checked)
            {
                Point point = (Point) Points[j];
                label6.Location = new Point(point.X, point.Y - 17);
                richTextBox6.Location = point;
                j = j + 1;
            }
        }
    }


    /// <summary>
    /// Submit the query to the desktop search, display results for selection and
start file watchers
    /// </summary>
    private void SearchBTN_Click(object sender, System.EventArgs e)
    {
        cleanupUI();

        //Adds to previous search history
        if (comboBox1.Items.Count > 0)
            if ((comboBox1.Text != comboBox1.Items[0].ToString()) && (comboBox1.Text !
```

```
    = ""))
            {
                string historyPath = Application.StartupPath + "\\Bin\\SearchHistory. ↙
    txt";
                try
                {
                    StreamWriter w = File.AppendText(historyPath);
                    w.Write(comboBox1.Text + "\r\n");
                    w.Close();
                }
                catch(Exception ex)
                {
                    throw new Exception(ex.Message);
                }
                loadSearchHistory();
            }

        //Submit query to MSN Desktop Search
        bool queryResult;
        string search;
        if (checkBox7.Checked)
            search = "title: " + comboBox1.Text;
        else
            search = comboBox1.Text;
        string filter = "document";
        queryResult = query.ExecuteQuery(search, filter);

        //Place the results into the ListBox so that it can be selected
        listBox1.Items.Clear();
        listBox2.Items.Clear(); //Remove any previous result to prevent confusion
        for (int i = 0; i < query.resultTable.Rows.Count; i++)
        {
            string fileName = query.resultTable.Rows[i].ItemArray[2].ToString(). ↙
    Substring(5); //File Name and location
            fileName = fileName.Replace('/', '\\');
            //Check if the file has an extension that can be read
            FileInfo fInfo = new FileInfo(fileName);
            string fext = fInfo.Extension.ToLower();
            if ((fext == ".html") || (fext == ".pdf") || (fext == ".doc"))
                listBox1.Items.Add(fileName);
        }

        //If there were no results found display a message
        if (query.resultTable.Rows.Count == 0)
            MessageBox.Show("  No results were found!\r\nTry checking your spelling\r\↙
    nor removing 'phrase must\r\n    be in the title' option");

        //Watch for the creation of the information file for processing
        textWatcher.Path         = Application.StartupPath + "\\Bin";
        textWatcher.Filter       = "file.txt";
        textWatcher.NotifyFilter = NotifyFilters.FileName |
            NotifyFilters.Attributes |
            NotifyFilters.LastAccess |
            NotifyFilters.LastWrite |
            NotifyFilters.Security |
            NotifyFilters.Size;

        textWatcher.Created += new FileSystemEventHandler(OnFileEventText);
        textWatcher.EnableRaisingEvents = true;

        //Watch for the creation of the view file for display
        rtfWatcher.Path         = Application.StartupPath + "\\Bin";
        rtfWatcher.Filter       = "file.rtf";
        rtfWatcher.NotifyFilter = NotifyFilters.FileName |
            NotifyFilters.Attributes |
            NotifyFilters.LastAccess |
            NotifyFilters.LastWrite |
            NotifyFilters.Security |
            NotifyFilters.Size;

        rtfWatcher.Created += new FileSystemEventHandler(OnFileEventRTF);
        rtfWatcher.EnableRaisingEvents = true;

        //For some reason the RichTextBox has to have been initiated for further files↙
```

```
     to be loaded on their creation
            ReadFileInfo(Application.StartupPath + "\\begin.rtf");
        }


        /// <summary>
        /// When a text file has been created from the search result file start processing
     and extracting information
        /// </summary>
        public void OnFileEventText(object source, FileSystemEventArgs fsea)
        {
            string filePath = Application.StartupPath + "\\Bin\\file.txt";
            progressBar1.Value = 60;

            //Introduce a delay
            if (fileType == ".html")
                Thread.Sleep(quickSleep);
            else
                Thread.Sleep(slowSleep);
            progressBar1.Value = 80;

            //Read information in the file to the Medicine class
            string info = "";
            Medicine med = new Medicine();

            //Read the entire contents of the file
            string infoFile = "";
            try
            {
                StreamReader reader = new StreamReader(filePath);
                infoFile = reader.ReadToEnd();
                reader.Close(); //Close the file once the contents are read
            }
            catch (Exception e)
            {
                throw new Exception(e.Message);
            }

            //Read through the file text and isolate the headings and mark them
            string[] line = new string[1000];
            int j=0;
            char endChar1 = ' ';
            char endChar2 = ' ';

            for (int i = 0; i < infoFile.Length; i++)
            {
                if((infoFile[i].ToString() == "\r") && (infoFile[i+1].ToString() == "\n"))
                {
                    //Try and determine if this new line is possibly a heading
                    if(j > 0)
                    {
                        //Determine the last character of the preceding line
                        if(line[j-1] != null)
                            endChar1 = line[j-1][line[j-1].Length - 1];
                        //Determine the last character of the line preceding the current
    by 2
                        if (j > 1)
                            if(line[j-2] != null)
                                endChar2 = line[j-2][line[j-2].Length - 1];

                        //Affix a header marker based on the following criteria
                        if((infoFile[i-1].ToString() != "") && (infoFile[i-1].ToString() !
    = ".")              //Its not an empty line or a sentence
                            && (infoFile[i-1].ToString() != ")") && (line[j] != null) &&
    (line[j-1] == null) //Its not bracketed and both current and previous lines are not
    null
                            && (endChar2 != ':') && (endChar2 != ';') && (endChar2 != '|')
                            //2 lines back is not a heading
                            && (endChar1 != ':') && (endChar1 != ';') && (endChar1 != '|
    '))
                                line[j]+= "|";  //Affix a heading marker
                    }
                    else
                        if (i > 0)
```

```
                        if((infoFile[i-1].ToString() != "") && (infoFile[i-1].ToString
() != ".")  //Its not an empty line or a sentence
                            && (infoFile[i-1].ToString() != ")") && (line[j] != null))
                //Its not bracketed and not null
                            line[j]+= "|";  //Affix a heading marker

                j++;    //increment line marker
                i++;    //increment character marker
            }
            else
                line[j]+= infoFile[i];  //Add the new character to the current line
        }

        //Read through the lines and watch for the heading markers to process the
information
        for (int k = 0; k < line.Length; k++)
        {
            string l = line[k];
            if (l != null)
            {
                string[] temp = l.Split('|');
                if(k > 0)
                {
                    if((l.IndexOf("|") > -1) && (temp[1].Length == 0) && (line[k-1] ==
 null)) //Make sure this is a line we should read
                    {
                        int found = ParseText(info, ref med);
                        info = "";
                    }
                }
                else
                {
                    if((l.IndexOf("|") > -1) && (temp[1].Length == 0)) //Make sure
this is a line we should read
                    {
                        int found = ParseText(info, ref med);
                        info = "";
                    }
                }
                char endChar3 = ' ';
                if (info != "")
                    endChar3 = info[info.Length - 1];
                if ((endChar3 != '|') && (info != ""))
                    info = info + "\r" + "\n" + l;
                else
                    info = info + l;
            }
        }

        //If at the end of the extraction process we do not find any interactions try
and see if it was listed under side effects
        if(med.Interactions == null)
        {
            int position = -1;
            if (med.SideEffects != null)
                position = med.SideEffects.ToLower().IndexOf("interactions", 0);
            if(position != -1)
            {
                med.Interactions = med.SideEffects.Substring(position + 12, med.
SideEffects.Length - position - 12).TrimStart(':',' ','-');
                med.SideEffects = med.SideEffects.Substring(0,position);
            }
        }

        //Perform an analysis on the File extraction and publish a report
        string analysis;
        analysis = "The information shown was extracted from: " + "\n" + fileTitle +"\
n\n";
        compTitle.Text = fileTitle;
        if(infoFile.ToLower().IndexOf("indication") > -1)
        {
            analysis += "- Indications appeared in the file";
            if(med.Indications != null)
                analysis += " and some relevant information was extracted";
```

```
            else
                analysis += " although no information could be extracted";
            analysis += ".\n";
        }
        else analysis += "- Indications could not be found!\n";
        if(infoFile.ToLower().IndexOf("contra-indication") > -1)
        {
            analysis += "- Contra-Indications appeared in the file";
            if(med.ContraIndications != null)
                analysis += " and some relevant information was extracted";
            else
                analysis += " although no information could be extracted";
            analysis += ".\n";
        }
        else analysis += "- Contra-Indications could not be found!\n";
        if((infoFile.ToLower().IndexOf("effect") > -1) && (infoFile.ToLower().IndexOf ↙
("side") > -1))
        {
            analysis += "- Side-effects appeared in the file";
            if(med.SideEffects != null)
                analysis += " and some relevant information was extracted";
            else
                analysis += " although no information could be extracted";
            analysis += ".\n";
        }
        else analysis += "- Side-effects could not be found!\n";
        if(infoFile.ToLower().IndexOf("dosage") > -1)
        {
            analysis += "- Dosage appeared in the file";
            if(med.Dosage != null)
                analysis += " and some relevant information was extracted";
            else
                analysis += " although no information could be extracted";
            analysis += ".\n";
        }
        else analysis += "- Dosage could not be found!\n";
        if(infoFile.ToLower().IndexOf("interaction") > -1)
        {
            analysis += "- Interactions appeared in the file";
            if(med.Interactions != null)
                analysis += " and some relevant information was extracted";
            else
                analysis += " although no information could be extracted";
            analysis += ".\n";
        }
        else analysis += "- Interactions could not be found!\n";
        if(infoFile.ToLower().IndexOf("warning") > -1)
        {
            analysis += "- Warnings appeared in the file";
            if(med.Warnings != null)
                analysis += " and some relevant information was extracted";
            else
                analysis += " although no information could be extracted";
            analysis += ".\n";
        }
        else analysis += "- Warnings could not be found!\n";
        if(infoFile.ToLower().IndexOf("composition") > -1)
        {
            analysis += "- Composition appeared in the file";
            if(med.Composition != null)
                analysis += " and some relevant information was extracted";
            else
                analysis += " although no information could be extracted";
            analysis += ".\n";
        }
        else analysis += "- Composition could not be found!\n";

        //Decompose the composition of the medicine into it components
        string[] ingred = {"","","","",""};
        string[] weight = {"","","","",""};
        string composition = "";

        int n = 0, p = 0;
        if (med.Composition != null)
```

```
            composition = med.Composition.ToString().ToLower().Substring(med.
    Composition.ToString().IndexOf("contain")+7,med.Composition.ToString().Length-med.
    Composition.ToString().IndexOf("contain")-7).TrimStart(' ','s');

            //Split the string into is compository words
            string[] comps = composition.Split(' ', '\r', '\n');

            //loop through the words found in the composition
            foreach (string c in comps)
            {
                string prior = "";
                string next = "";

                //Determine if numbers exist in the string
                MatchCollection matches = Regex.Matches(c, @"\d+", RegexOptions.Compiled);

                //If a number is found try and see where the matching ingredient is
                if (matches.Count >= 1)
                {
                    //First determine which word precedes and comes after the weight
                    if (n > 0)
                        prior = comps[n-1].TrimEnd('.', ',');
                    if (n+1 < comps.Length)
                    {
                        next = comps[n+1].TrimEnd('.', ',');
                        if ((next.Length <= 2) && (n+2 < comps.Length))
                            next = comps[n+2].TrimEnd('.', ',');
                    }

                    //Extract the weight
                    weight[p] = c.TrimEnd('.', ',');

                    //Extract the ingredient
                    if (((prior.TrimEnd('s') != "contain") && (prior.Length > 3)))
                        ingred[p] = prior;
                    else
                        ingred[p] = next;

                    if (p < 4) p++; //Search for upto 5 ingredients
                }
                n++; //Keep track of the word in the composition
            }

            //Reset Generic Query
            genericQuery.Text = "";
            checkBox8.Checked = false;
            checkBox9.Checked = false;

            //Display extracted composition information for selection
            checkBox8.Text = ingred[0];
            if(checkBox8.Text != "") checkBox8.Visible = true;
            else checkBox8.Visible = false;
            checkBox9.Text = weight[0];
            if(checkBox9.Text != "") checkBox9.Visible = true;
            else checkBox9.Visible = false;

            //Display a report of the generic retrieval process
            richTextBox7.Text = analysis;

            //Display the information contained in the Medicine class
            richTextBox1.Text = med.ContraIndications;
            richTextBox2.Text = med.Indications;
            richTextBox3.Text = med.SideEffects;
            richTextBox4.Text = med.Dosage;
            richTextBox5.Text = med.Interactions;
            richTextBox6.Text = med.Warnings;
            richTextBox8.Text = med.Composition;
            PositionResults();

            progressBar1.Value = 100;
        }


        /// <summary>
```

```
    /// When a richt text file has been created from the search result display it in the view tab
    /// </summary>
    public void OnFileEventRTF(object source, FileSystemEventArgs fsea)
    {
        if (fileType == ".html")
            Thread.Sleep(quickSleep);
        else
            Thread.Sleep(slowSleep);
        try
        {
            //Display the File for view by the Doctor
            ReadFileInfo(Application.StartupPath + "\\Bin\\file.rtf");
            fileLoadArea.ZoomFactor = 1.5f; //Increase the font size to make it easier to read
            fileLoadArea.Select();
        }
        catch (Exception e)
        {
            throw new Exception(e.Message);
        }
    }


    /// <summary>
    /// When a search result is selected transform it into a text file for the system to process
    /// </summary>
    private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
    {
        cleanupUI();

        //Delete old information contained in the respective files
        File.Delete(Application.StartupPath + "\\Bin\\file.txt");
        File.Delete(Application.StartupPath + "\\Bin\\file.rtf");

        string fileName = listBox1.SelectedItem.ToString();
        FileInfo fInfo = new FileInfo(fileName);
        //Determine the type of the file before displaying the contents
        string fext = fInfo.Extension.ToLower();
        fileType = fext;
        string arg = "";
        progressBar1.Value = 10;

        //Build a command to create a text file for processing
        switch(fext)
        {
            case ".html":
                arg = " /S " + fileName + " /F4 /T " + Application.StartupPath + "\\Bin\\file.txt" + " /C1 /M2";
                break;
            case ".pdf":
                arg = " /S " + fileName + " /T " + Application.StartupPath + "\\Bin\\file.txt" + " /C1 /M3 /E";
                break;
            case ".doc":
                arg = " /S " + fileName + " /F9 /T " + Application.StartupPath + "\\Bin\\file.txt" + " /C1 /M2";
                break;
        }
        System.Diagnostics.Process.Start("D:\\Program Files\\Softinterface, Inc\\Convert Doc\\ConvertDoc.exe", arg);

        //Build a command to create a Rich Text File for viewing purposes
        switch(fext)
        {
            case ".html":
                arg = " /S " + fileName + " /F4 /T " + Application.StartupPath + "\\Bin\\file.rtf" + " /C5 /M2";
                break;
            case ".pdf":
                arg = " /S " + fileName + " /T " + Application.StartupPath + "\\Bin\\file.rtf" + " /C3 /M3 /E";
                break;
```

```
            case ".doc":
                arg = " /S " + fileName + " /F9 /T " + Application.StartupPath + "\\
Bin\\file.rtf" + " /C5 /M2";
                break;
        }
        System.Diagnostics.Process.Start("D:\\Program Files\\Softinterface, Inc\\
Convert Doc\\ConvertDoc.exe", arg);

        //Extract the document title
        fileTitle = query.resultTable.Rows[listBox1.SelectedIndex].ItemArray[0].
ToString();
        label7.Text = "File: " + listBox1.SelectedItem.ToString();
        progressBar1.Value = 40;
    }


    /// <summary>
    /// When a generic result is selected transform it into a text file for the system
 to process
    /// </summary>
    private void listBox2_SelectedIndexChanged(object sender, System.EventArgs e)
    {
        cleanupUI();

        //Delete old information contained in the respective files
        File.Delete(Application.StartupPath + "\\Bin\\file.txt");
        File.Delete(Application.StartupPath + "\\Bin\\file.rtf");

        string fileName = listBox2.SelectedItem.ToString();
        FileInfo fInfo = new FileInfo(fileName);
        //Determine the type of the file before displaying the contents
        string fext = fInfo.Extension.ToLower();
        fileType = fext;
        string arg = "";

        //Build a command to create a text file for processing
        switch(fext)
        {
            case ".html":
                arg = " /S " + fileName + " /F4 /T " + Application.StartupPath + "\\
Bin\\file.txt" + " /C1 /M2";
                break;
            case ".pdf":
                arg = " /S " + fileName + " /T " + Application.StartupPath + "\\Bin\\
file.txt" + " /C1 /M3 /E";
                break;
            case ".doc":
                arg = " /S " + fileName + " /F9 /T " + Application.StartupPath + "\\
Bin\\file.txt" + " /C1 /M2";
                break;
        }
        System.Diagnostics.Process.Start("D:\\Program Files\\Softinterface, Inc\\
Convert Doc\\ConvertDoc.exe", arg);

        //Build a command to create a Rich Text File for viewing purposes
        switch(fext)
        {
            case ".html":
                arg = " /S " + fileName + " /F4 /T " + Application.StartupPath + "\\
Bin\\file.rtf" + " /C5 /M2";
                break;
            case ".pdf":
                arg = " /S " + fileName + " /T " + Application.StartupPath + "\\Bin\\
file.rtf" + " /C3 /M3 /E";
                break;
            case ".doc":
                arg = " /S " + fileName + " /F9 /T " + Application.StartupPath + "\\
Bin\\file.rtf" + " /C5 /M2";
                break;
        }
        System.Diagnostics.Process.Start("D:\\Program Files\\Softinterface, Inc\\
Convert Doc\\ConvertDoc.exe", arg);

        //Extract the document title
```

```
            fileTitle = listBox2.SelectedItem.ToString();
            label7.Text = "File: " + listBox2.SelectedItem.ToString();
        }


        /// <summary>
        /// When the Generic Search is clicked submit query to search engine and process returned results to ensure correct generic
        /// </summary>
        private void genericQueryBtn_Click(object sender, System.EventArgs e)
        {
            if (genericQuery.Text != "")
            {
                cleanupUI();

                //Submit query to MSN Desktop Search
                bool queryResult;
                string filter = "document";
                queryResult = query.ExecuteQuery(genericQuery.Text, filter);

                //Place the results into the ListBox so that it can be selected
                listBox2.Items.Clear();
                listBox1.Items.Clear();

                //Start a thread to process returned results and appropriate ones to the results listbox
                Extractor extract = new Extractor(listBox2, query, genericQuery.Text, richTextBox9, progressBar2, button1);
                backgroundThread = new Thread(new ThreadStart(extract.findGenerics));
                backgroundThread.Start();
                button1.Visible = true;
            }
            else
                MessageBox.Show("No query has been entered!");  //Trap an empty query
        }


        /// <summary>
        /// Implements functionality to find text in the View Tab
        /// </summary>
        private void FindBTN_Click(object sender, System.EventArgs e)
        {
            // Create a string to search for the word
            String searchString = textBox1.Text.ToLower();
            // Determine the starting location of the word
            indexFound = fileLoadArea.Text.ToLower().IndexOf(searchString, indexFound+1);
            // Determine if the word has been found and select it if it was.
            if (indexFound != -1)
            {
                //Select the RichTextBox
                fileLoadArea.Select();
                // Select the string using the index and the length of the string.
                fileLoadArea.Select(indexFound, searchString.Length);
                fileLoadArea.ScrollToCaret();   //Move the window to where the occurence was found
            }
            else
                MessageBox.Show("No further occurences were found!");
        }


        /// <summary>
        /// When the text to be found is changed the marker must be reset
        /// </summary>
        private void textBox1_TextChanged(object sender, System.EventArgs e)
        {
            //Reset the position marker for the find function when a new Find phrase is used
            indexFound = 0;
        }


        /// <summary>
        /// Cleanup the UI to avoid confusion
```

```
        /// </summary>
        private void cleanupUI()
        {
            //Delete the analysis information
            richTextBox1.Text = "";
            richTextBox2.Text = "";
            richTextBox3.Text = "";
            richTextBox4.Text = "";
            richTextBox5.Text = "";
            richTextBox6.Text = "";
            richTextBox7.Text = "";
            richTextBox8.Text = "";
            try
            {
                if (backgroundThread.ThreadState != 0)
                {
                    richTextBox9.Text = "";
                    progressBar2.Value = 0;
                }
            }
            catch
            {
                richTextBox9.Text = "";
                progressBar2.Value = 0;
            }

            //reset the options
            label7.Text = "";
            compTitle.Text = "";
            fileLoadArea.Clear();
            checkBox8.Visible = false;
            checkBox9.Visible = false;
            progressBar1.Value = 0;
        }


        /// <summary>
        /// Loads the history of previously entered search phrases
        /// </summary>
        private void loadSearchHistory()
        {
            string historyPath = Application.StartupPath + "\\Bin\\SearchHistory.txt";
            int count = 0;

            try
            {
                StreamReader reader = new StreamReader(historyPath);
                string historyFile = reader.ReadToEnd();
                string [] history = historyFile.Split('\n');
                comboBox1.Items.Clear();    //Clear the current history

                //Go to the end and start adding queries from the end of the file till 10 ↙
    queries have been added
                for (int i = history.Length - 1; ((count < 11) && (i >= 0)); i--)
                {
                    if (history[i].TrimEnd('\r') != "")
                        comboBox1.Items.Add(history[i].TrimEnd('\r'));
                    count++;
                }
                reader.Close();
            }
            catch
            {
                StreamWriter w = new StreamWriter(historyPath);
                w.Write("Aspirin\r\n");
                w.Close();
            }
        }

        /// <summary>
        /// Cancel the generic search
        /// </summary>
        private void button1_Click(object sender, System.EventArgs e)
        {
```

```
            backgroundThread.Abort();
            button1.Visible = false;
            richTextBox9.Text += "\r\nSearch was cancelled!";
        }


        /// <summary>
        /// Upon exit close window
        /// </summary>
        private void menuItem4_Click(object sender, System.EventArgs e)
        {
            this.Close();
        }


        /// <summary>
        /// Upon program being closed abort threads and exit
        /// </summary>
        private void Search_Closing(object sender, System.ComponentModel.CancelEventArgs ↙
    e)
        {
            try
            {
                backgroundThread.Abort();
            }
            catch
            {
            }
            Environment.Exit(0);
        }


        #region Place information
        /// <summary>
        /// Display inforamtion according to selections made
        /// </summary>
        private void checkBox1_CheckedChanged(object sender, System.EventArgs e)
        {
            if(checkBox1.CheckState == CheckState.Checked)
            {
                checkBoxCounter = checkBoxCounter + 1;
                richTextBox2.Visible = bool.Parse("True");
                label1.Visible = bool.Parse("True");
            }
            else
            {
                checkBoxCounter = checkBoxCounter - 1;
                richTextBox2.Visible = bool.Parse("False");
                label1.Visible = bool.Parse("False");
            }
            PositionResults();
        }

        private void checkBox2_CheckedChanged(object sender, System.EventArgs e)
        {
            if(checkBox2.CheckState == CheckState.Checked)
            {
                checkBoxCounter = checkBoxCounter + 1;
                richTextBox1.Visible = bool.Parse("True");
                label2.Visible = bool.Parse("True");
            }
            else
            {
                checkBoxCounter = checkBoxCounter - 1;
                richTextBox1.Visible = bool.Parse("False");
                label2.Visible = bool.Parse("False");
            }
            PositionResults();
        }

        private void checkBox3_CheckedChanged(object sender, System.EventArgs e)
        {
            if(checkBox3.CheckState == CheckState.Checked)
            {
```

```
            checkBoxCounter = checkBoxCounter + 1;
            richTextBox4.Visible = bool.Parse("True");
            label4.Visible = bool.Parse("True");
        }
        else
        {
            checkBoxCounter = checkBoxCounter - 1;
            richTextBox4.Visible = bool.Parse("False");
            label4.Visible = bool.Parse("False");
        }
        PositionResults();
    }

    private void checkBox4_CheckedChanged(object sender, System.EventArgs e)
    {
        if(checkBox4.CheckState == CheckState.Checked)
        {
            checkBoxCounter = checkBoxCounter + 1;
            richTextBox5.Visible = bool.Parse("True");
            label5.Visible = bool.Parse("True");
        }
        else
        {
            checkBoxCounter = checkBoxCounter - 1;
            richTextBox5.Visible = bool.Parse("False");
            label5.Visible = bool.Parse("False");
        }
        PositionResults();
    }

    private void checkBox5_CheckedChanged(object sender, System.EventArgs e)
    {
        if(checkBox5.CheckState == CheckState.Checked)
        {
            checkBoxCounter = checkBoxCounter + 1;
            richTextBox3.Visible = bool.Parse("True");
            label3.Visible = bool.Parse("True");
        }
        else
        {
            checkBoxCounter = checkBoxCounter - 1;
            richTextBox3.Visible = bool.Parse("False");
            label3.Visible = bool.Parse("False");
        }
        PositionResults();
    }

    private void checkBox6_CheckedChanged(object sender, System.EventArgs e)
    {
        if(checkBox6.CheckState == CheckState.Checked)
        {
            checkBoxCounter = checkBoxCounter + 1;
            richTextBox6.Visible = bool.Parse("True");
            label6.Visible = bool.Parse("True");
        }
        else
        {
            checkBoxCounter = checkBoxCounter - 1;
            richTextBox6.Visible = bool.Parse("False");
            label6.Visible = bool.Parse("False");
        }
        PositionResults();
    }
    #endregion

    #region Place Composition components
    //Display Weights and Ingredients found in the Composition
    private void checkBox8_CheckedChanged(object sender, System.EventArgs e)
    {
        if (checkBox8.Checked == true)
            genericQuery.Text+= " " + checkBox8.Text;
        else
            if (genericQuery.Text.IndexOf(checkBox8.Text) != -1)
            genericQuery.Text = genericQuery.Text.Remove(genericQuery.Text.IndexOf
```

```
    (checkBox8.Text), checkBox8.Text.Length);
            genericQuery.Text = genericQuery.Text.TrimStart(null);
        }

        private void checkBox9_CheckedChanged(object sender, System.EventArgs e)
        {
            if (checkBox9.Checked == true)
                genericQuery.Text+= " " + checkBox9.Text;
            else
                if (genericQuery.Text.IndexOf(checkBox9.Text) != -1)
                genericQuery.Text = genericQuery.Text.Remove(genericQuery.Text.IndexOf
    (checkBox9.Text), checkBox9.Text.Length);
            genericQuery.Text = genericQuery.Text.TrimStart(null);
        }
        #endregion

    }
}
```

```
using System;

namespace PharmAide
{
    /// <summary>
    /// Medicine is the class that stores all the relevant attributes of a particular
    medication
    /// </summary>
    public class Medicine
    {
        private string composition;
        private string indications;
        private string contraIndications;
        private string dosage;
        private string sideEffects;
        private string warnings;
        private string interactions;

        public Medicine()
        {
        }

        public string Composition
        {
            get
            {
                return this.composition;
            }
            set
            {
                this.composition = value;
            }
        }

        public string Indications
        {
            get
            {
                return this.indications;
            }
            set
            {
                this.indications = value;
            }
        }

        public string ContraIndications
        {
            get
            {
                return this.contraIndications;
            }
            set
            {
                this.contraIndications = value;
            }
        }

        public string Dosage
        {
            get
            {
                return this.dosage;
            }
            set
            {
                this.dosage = value;
            }
        }

        public string SideEffects
        {
            get
            {
```

```
                return this.sideEffects;
            }
            set
            {
                this.sideEffects = value;
            }
        }

        public string Warnings
        {
            get
            {
                return this.warnings;
            }
            set
            {
                this.warnings = value;
            }
        }

        public string Interactions
        {
            get
            {
                return this.interactions;
            }
            set
            {
                this.interactions = value;
            }
        }
    }
}
```

```
using System;
using System.Windows.Forms;
using System.IO;
using System.Threading;

namespace PharmAide
{
    /// <summary>
    /// Extraxtor is used to determine suitable matches for generics.
    /// </summary>
    public class Extractor
    {
        // State information used in the task.
        private ListBox listbox;
        private QueryBuilder query;
        private string genericQuery;
        private RichTextBox result;
        private ProgressBar progressBar;
        private Button button;

        public Extractor(ListBox listboxT, QueryBuilder queryT, string genericQueryT,
    RichTextBox resultT, ProgressBar progressBarT, Button buttonT)
        {
            listbox = listboxT;
            query = queryT;
            genericQuery = genericQueryT;
            result = resultT;
            progressBar = progressBarT;
            button = buttonT;
        }

        public void findGenerics()
        {
            progressBar.Value = 0;
            //Define the path where the generic information is going to be stored
            string genericFilePath = Application.StartupPath + "\\Bin\\scan.txt";

            ListBox l = listbox;
            result.Text += query.resultTable.Rows.Count.ToString() + " results were
    retrieved from the search engine.\r\n";

            //Go through the results returned and determine if the composition matches the
     query
            for (int i = 0; i < query.resultTable.Rows.Count; i++)
            {
                //Delete the previously processed file
                try
                {
                    File.Delete(genericFilePath);
                }
                catch
                {
                }

                //Get File Name and location from listbox
                string fileName = query.resultTable.Rows[i].ItemArray[2].ToString().
    Substring(5);
                fileName = fileName.Replace('/', '\\');
                FileInfo fInfo = new FileInfo(fileName);
                //Determine the type of the file before displaying the contents
                string fext = fInfo.Extension.ToLower();
                string arg = "";

                //Build a command to create a text file for processing
                switch(fext)
                {
                    case ".html":
                        arg = " /S " + fileName + " /F4 /T " + genericFilePath + " /C1 /M2
    ";
                        break;
                    case ".pdf":
                        arg = " /S " + fileName + " /T " + genericFilePath + " /C1 /M3 /E"
    ;
                        break;
```

```
                    case ".doc":
                        arg = " /S " + fileName + " /F9 /T " + genericFilePath + " /C1 /M2↙
    ";
                        break;
                }
                System.Diagnostics.Process.Start("D:\\Program Files\\Softinterface, Inc\\ ↙
    Convert Doc\\ConvertDoc.exe", arg);

                //Wait for the the file to be created before processing it
                while (File.Exists(genericFilePath) != true)
                {
                    if (fext == ".html")
                        Thread.Sleep(400);
                    else
                        Thread.Sleep(2000);
                }

                //Read the entire contents of the file
                string genericFile = "";
                try
                {
                    StreamReader reader = new StreamReader(genericFilePath);
                    genericFile = reader.ReadToEnd();
                    reader.Close(); //Close the file once the contents are read
                }
                catch (Exception e)
                {
                    throw new Exception(e.Message);
                }

                //Extract the composition from the file
                string composition = "";
                if ((genericFile.ToLower().IndexOf("composition", 200) + 200 <=           ↙
    genericFile.Length) && genericFile.ToLower().IndexOf("composition", 200) != -1)
                    composition = genericFile.Substring(genericFile.ToLower().IndexOf(    ↙
    "composition", 200), 200).ToLower();

                //Find the elements of the composition and use that to match generics
                string[] word = genericQuery.Split(' ');
                int found = 1;
                foreach (string w in word)
                {
                    if (composition != null)
                        if (composition.IndexOf(w.ToLower()) == -1)
                            found = 0;
                }
                if ((found == 1) && ((fext == ".html") || (fext == ".pdf") || (fext == ". ↙
    doc")))
                    l.Items.Add(fileName);

                if (fext == ".pdf")
                    Console.WriteLine("Hello");

                //Publish a report
                result.Text += "\r\nFile " + i.ToString() + ", " + fileName.ToString() + "↙
     has been processed.";
                progressBar.Value = ((i + 1) * 100) / query.resultTable.Rows.Count;
            }
            result.Text += "\r\n\r\nSearch results have been filtered to include only     ↙
    those with matching composition.\r\nGenerics were found with composition matching     ↙
    query";
            MessageBox.Show("Search Complete!");
            button.Visible = false;
        }
    }
}
```